Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jernej Hartman

Razvoj sistema za podporo storitev v oblaku

magistrsko delo

študijski program druge stopnje
računalništvo in informatika

Mentor: dr. Andrej Brodnik
Somentor: prof. dr. Matjaž Branko Jurič

Ljubljana, 2016

Zahvaljujem se dr. Andreju Brodnikus za vse strokovne nasvete in pomoč med pisanjem magistrske naloge.
Iskrena zahvala gre tudi družini, punci Urši in najbližjim za podporo med vsemi leti študija. Hvala tudi vsem ostalim, ki ste mi med študijem stali ob strani in me podpirali.

Kazalo

Kazalo slik

Seznam uporabljenih kratic

Kratica	Angleško	Slovensko
API	Application programming interface	programski vmesnik
SDK	Software development kit	razvojna orodja za razvoj programske opreme
BaaS	Backend as a service	podporni sistem kot storitev
PaaS	Platform as a service	platforma kot storitev
SaaS	Software as a service	programska oprema kot storitev
VMM	Virtual machine manager	nadzornik navideznih naprav
VM	Virtual machine	navidezna naprava
HTTP	Hypertext transfer protocol	protokol za izmenjavo hiperteksta
JSON	JavaScript object notation	format zapisa podatkov v ključ-vrednost obliki
CRM	Customer relationship managment	sistem za upravljanje odnosov s strankami
DBaaS	Database as a service	podatkovna baza kot storitev
STaaS	Storage as a service	shranjevanje podatkov kot storitev
SECCaaS	Security as a service	varnost kot storitev
DaaS	Desktop as a service	namizje kot storitev
HTML	Hypertext markup language	markirni jezik za hipertekst
MBaaS	Mobile backend as a service	mobilni podporni sistem kot storitev
TCP	Transmission Control Protocol	protokol za krmiljenje prenosa
AES	Advanced Encryption Standard	napredni standard za šifriranje
TLS	Transport Layer Security	varnostni transportni sloj

Povzetek

S porastom računalništva v oblaku se razvija vedno več sistemov, ki rešujejo različne težave, na katere naletijo razvijalci. Za potrebe mobilnih aplikacij so se razvili podporni sistemi, ki omogočajo izvajanje različnih storitev na oddaljenih strežnikih. S takim načinom delovanja se razvijalci izognejo vključitvi potrebne programske opreme storitev v aplikacijo, kar posledično skrajša razvojni čas. Za dostop do tovrstnih sistemov aplikacije potrebujejo dostopne ključe, ki jih shranijo v izvorno kodo aplikacije. Če se aplikacije izvajajo na napravah, do katerih ima uporabnik dostop, lahko to privede do nepooblaščenega dostopa in razkritja podatkov. V magistrskemu delu predstavimo zasnovo in razvoj novega podpornega sistema, ki izboljšuje obstoječe sisteme na področju varnosti, izolacije in zaščite podatkov.

Ključne besede

Računalništvo v oblaku, varnost in zasebnost v oblaku, sistem za podporo storitev v oblaku

Abstract

Cloud computing is becoming increasingly important and a growing number of systems are being developed to help solve different problems that the developers are facing. Numerous systems for enabling different backend services on remote servers have been developed for mobile applications. By doing so the developers do not have to include the necessary software in the application, which results in a shorter development time. To access these systems, applications need access keys, saved in the application's source code. If the applications run on the devices that the user has access to, it can lead to an unauthorised access and disclosure of information.
This paper presents the outline and the development of a new backend as a service system which improves the existing systems in the field of data security, isolation and protection.

Keywords
Cloud computing, security and privacy in the cloud, backend as a service

Poglavje 1

Uvod

Razvoj spletnih aplikacij, ki so namenjene veliko uporabnikom, nikoli ni bila lahka naloga. Pri razvoju tovrstnih aplikacij je treba predvideti različne odpovedi in omejitve programske in strojne opreme. V preteklosti je to pomenilo prilagoditev in optimizacijo vsake komponente aplikacije posebej, kar pa je bil za začetek razvoja velik strošek [1, 2]. Rešitve takih problemov so bile prilagojene vsaki aplikaciji posebej in so se razlikovale med organizacijami. Podjetja so morala za primer povečanega povpraševanja vnaprej zakupiti večje količine strojne opreme, ki jo je bilo treba hraniti v primernem prostoru in predvideti različne izpade in okvare opreme. Proces postavitve in nadzorovanja takih sistemov je bil počasen, kompleksen in zamuden. Z razvojem računalništva v oblaku se je to spremenilo, saj so se pojavili ponudniki različnih storitev, ki razvijalcem na oddaljenih napravah omogočajo izvajanje aplikacij in storitev preko omrežja [1–5]. Razvijalcem se ni več treba obremenjevati z vso strojno in programsko opremo, ki je potrebna za nadzorovanje in upravljanje sistemov. Storitve računalništva v oblaku omogočajo postavitev aplikacij na infrastrukturo ponudnika, ki nudi vso potrebno strojno in programsko opremo za delovanje aplikacije, hkrati pa poskrbi za nadgradnje vse opreme [1,3,6,7]. To omogoča visoko zanesljivost delovanja aplikacij z minimalnim finančnim vložkom.

Mobilne in spletne aplikacije morajo shranjevati in procesirati različne oblike podatkov, ki se lahko nahajajo lokalno na napravah ali pa na oddaljenih strežnikih oziroma v oblaku. Da se to doseže, mora aplikacija vsebovati različne programske

knjižnice, ki ji to omogočajo. Proces razvoja, postavitve in nadzorovanja tovrstne programske opreme oziroma storitev je za razvijalce zamuden, saj morajo imeti znanja z različnih področij razvoja programske opreme [8–10]. Vsako storitev je treba postaviti, po potrebi tudi razviti, predvideti vse mogoče odpovedi, poskrbeti za visoko dostopnost storitve in jo nato še integrirati v aplikacijo. Za odpravo navedenih pomanjkljivosti se je razvil nov model računalništva v oblaku – BaaS (angl. *backend as a service*).

Sistemi BaaS razvijalcem aplikacij omogočajo uporabo različnih storitev, ki se izvajajo na oddaljenih strežnikih, preko enotnega vmesnika [8,11,12]. S tem odpravijo potrebo po postavitvi strežnikov, po razvoju in upravljanju posamezne storitve in integraciji vsake posamezne storitve v aplikacijo. Vse to skrajša razvojni čas in finančni vložek, potreben za razvoj. Med najpomembnejše storitve spadajo shranjevanje različnih oblik podatkov, overjanje in identifikacija uporabnikov ter njihovih sej.

1.1 Cilji

V magistrskem delu sta predstavljena zasnova in razvoj novega podpornega sistema s poudarkom na varnosti in zaščiti podatkov. Podatke bomo zavarovali s pomočjo simetričnih in asimetričnih kriptografskih algoritmov in tako zavarovali podatke sistemov, postavljenih na javnih oblakih in/ali storitvah. Glavna pomanjkljivost obstoječih sistemov je uporaba njihovih storitev na napravah, nad katerimi imajo uporabniki popoln nadzor. Za odpravo tovrstnih težav nekateri ponudniki omogočajo uporabo storitev le ob prisotnosti prijavljenega uporabnika [8]. To razvijalcem aplikacij, ki nimajo potreb po prijavnem sistemu, otežuje uporabo sistema in jih sili v shranjevanje prijavnih podatkov v izvorno kodo aplikacije. Ker lahko to pripelje do razkritja dostopnih ključev in izgube podatkov [13–15], bo dostop do sistema mogoč le preko strežniških aplikacij, nad katerimi imajo nadzor razvijalci. Sistemi BaaS že v osnovi nudijo večnajemniško arhitekturo, za to bo potrebna vpeljava izolacije podatkov. Ta bo dosežena s pomočjo šifriranja, tako tudi v primeru nepooblaščenega dostopa do strežnika ne more priti do razkritja podatkov.

1.2 Struktura magistrske naloge

V prvem poglavju magistrske naloge sta s pomočjo namestitvenih in storitvenih modelov predstavljena računalništvo v oblaku in njegova celotna struktura. Ti modeli predstavljajo vse mogoče načine namestitve oblakov in različnih tipov storitev, ki jih oblaki nudijo.

Drugo poglavje predstavlja zasnovo sistema BaaS, ki je cilj te magistrske naloge. Zasnova se začne z predstavitvijo motivacije magistrskega dela, ki na primeru preproste aplikacije izpostavi vse prednosti in slabosti tovrstnih sistemov. Temu sledita definicija zahtev sistema in opis vseh njenih komponent. Komponente prikazujejo način doseganja izoliranosti in varnosti podatkov, delovanje prijavnega sistema za identifikacijo entitet in način implementacijo storitev sistema.

Tretje poglavje obravnava implementacijo in razvoj sistema, ki smo ga zasnovali v drugem poglavju. Predstavljeni so realizacija vseh komponent sistema, kar vključuje protokol, po katerem se sporočila med entitetami izmenjujejo, struktura vseh podatkovnih baz, definicija programskega vmesnika storitev in opis osnovnih storitev sistema.

Četrto poglavje predstavlja uporabo razvitega sistema s pomočjo razvoja testne aplikacije. Uporaba prikazuje registracijo aplikacije v sistem, definicijo njenih storitev, strukturo njene konfiguracijske datoteke in prikaz uporabe ene izmed njenih storitev. Za konec razviti sistem primerjamo z obstoječimi sistemi BaaS.

V zadnjem poglavju predstavimo sklepne ugotovitve magistrskega dela.

Poglavje 2

Računalništvo v oblaku

Računalništvo v oblaku je širok in sorazmerno nov pojem, ki ga je težko opredeliti, saj se njegova definicija razlikuje celo med strokovnjaki in predstavniki v istih strokah. Kljub novosti mnoge značilnosti računalništva v oblaku niso nove, saj se oblačne storitve v praksi uporablja že dlje.
Uporaba storitev oblaka je najbolj razširjena v podjetjih, ki morajo zagotavljati nemoteno delovanje svojih produktov ali storitev, dostopnih preko spleta [2,4]. To je lahko preprosta spletna stran ali pa napreden sistem za vodenje podjetja. Tudi kadar se povpraševanje po storitvi poveča, mora biti storitev dostopna, kar pa zahteva večje kapacitete strojne opreme. Za to je bilo v preteklosti treba zakupiti večje kapacitete strežnikov, kar pa je pomenilo povečanje finančnih vložkov, ki so potrebni za začetek razvoja [4, 7].
Podjetja, kot so Google, Yahoo, Amazon in Microsoft, so za potrebe svojih aplikacij razvila lastne arhitekture, ki rešujejo težave pri zagotavljanju visoke zanesljivosti delovanja, svojo tehnologijo pa so delila z razvijalci [2,3]. S pomočjo storitev oblaka lahko razvijalci svoje inovativne ideje uresničujejo brez velikih finančnih vložkov v strojno opremo in človeške vire.

Računalništvo v oblaku v veliki večini predstavljajo aplikacije in storitve, ki se izvajajo na navideznih virih in so dostopne odjemalcem preko omrežja, najpogosteje preko interneta [3, 4]. S pomočjo namenskih aplikacij ponudniki oblačnih storitev omogočajo upravljanje strojnih in/ali programskih virov brez vmesnega človeškega faktorja, to pa zaračunajo po porabi in ne po vnaprejšnjem zakupu [2, 3, 7].

Osnovne značilnosti računalništva v oblaku, kot jih definira NIST (angl. *National Institute of Standards and Technology*), so [16]:

storitev na zahtevo (angl. *on-demand self-service*): uporabniki storitve glede na trenutne potrebe prilagajajo računalniške zmogljivosti, ki vključujejo procesno moč, velikost pomnilnika, velikost shrambe in pasovno širino povezave brez interakcije s fizičnimi osebami ponudnika storitve,

širok dostop do omrežja (angl. *broad network access*): storitve računalništva v oblaku so dostopne preko splošnih internetnih protokolov in omrežnih standardov, do njih dostopamo iz različnih odjemalcev, ki se lahko izvajajo na različnih napravah – osebnih računalnikih, notesnikih, mobilnih in namenskih napravah,

združevanje virov (angl. *resource pooling*): viri ponudnika so združeni in so na voljo vsem uporabnikom storitve. Uporabniku se, glede na njegove potrebe, fizični in navidezni viri dinamično dodeljujejo, ne da bi poznali njihovo natančno lokacijo,

hitra elastičnost (angl. *rapid elasticity*): računalniške zmogljivosti se dinamično zasedajo in sproščajo glede na trenutno obremenitev in potrebe uporabnikov. Uporabniku so viri vedno na voljo in zanj delujejo neomejeno,

merljivost (angl. *measured service*): vsaka uporaba virov je merljiva, tako poraba strojne opreme (centralne procesne enote, pomnilnika, trdega diska, pasovne širine) kot tudi programske opreme (velikost in količina podatkov v podatkovnih bazah, količina podatkov, prejeta preko omrežja ipd.).

Ponudniki svoje vire nudijo kot storitev (angl. *as a service*), do katere razvijalci dostopajo preko programskih vmesnikov API [2, 3, 17]. S takim delovanjem se ponudniki osredotočijo na zagotavljanje visoke zanesljivosti delovanja in dostopnosti svoje storitve, neodvisno od uporabnikov storitve. Uporaba oblačnih sistemov se razlikuje med ponudniki, za končne uporabnike aplikacij pa je ta uporaba transparentna. Do njih lahko dostopajo na več načinov, najpogostejša sta preko spletnega brskalnika ali namenske aplikacije, ki se izvaja na osebnem računalniku, mobilniku ali na namenski napravi.

2.1 Prednosti in slabosti oblaka

Tako kot pri vseh sistemih in storitvah se tudi pri računalništvu v oblaku srečujemo s prednostmi in slabostmi. Prednosti so [1–3, 6, 16, 17]:

- namesto zakupa vse potrebne strojne in programske opreme lahko uporabnik opremo najame od ponudnika oblaka in najem plača glede na dejansko porabo virov. To skrajša razvojni čas in zniža stroške,
- hitra elastičnost uporabniku omogoča, da v primeru povišane obremenitve prilagodi računske zmogljivosti in tako zagotovi nemoteno delovanje storitve, in
- uporabnik lahko prilagaja računske vire brez potreb po interakciji s fizičnimi osebami ponudnika storitve, kar omogoča avtomatizacijo celotnega procesa v primeru povišane obremenitve, kar posledično omogoča izdelavo robustnejših storitev.

Poleg navedenih prednosti so se v praksi pokazale tudi naslednje slabosti [1–3, 5, 7, 17–19]:

- uporaba storitev v oblaku privede do skrbi glede varnosti, ki so specifične za računalništvo v oblaku. To vključuje napade DDoS (angl. *distributed denial of service*), ki zlonamerno povečajo porabo. Hitra elastičnost in združevanje virov omogočata, da se viri sproščajo in dodeljujejo, kar omogoča obnovitev podatkov iz sproščenih virov. Večina storitev oblaka se zanaša na virtualizacijo in v primeru njene ranljivosti lahko napadalec iz navideznega okolja pridobi dostop do celotnega sistema (angl. *virtual machine escape*). Večina aplikacij in programskih vmesnikov je dostopna preko protokola HTTP, ki ne shranjuje stanj (angl. *stateless*), to pa lahko privede do ugrabitve seje (angl. *session hijacking*). Kot slabost se kažejo tudi ranljivosti v namenskih aplikacijah za upravljanje virov – napadi z vrivanjem SQL,
- uporabniki morajo ponudniku storitve zaupati, saj ne vedo, kako so podatki zaščiteni in kdo ima do njih dostop. To pripelje do težav z zaupnostjo, saj si s ponudnikom delimo odgovornost glede zaupnosti in varovanja podatkov,

– ponudba oblačnih storitev se razlikuje med ponudniki (ni enotnega standarda za uporabo tovrstnih storitev), izbira ponudnika lahko pripelje do precej oteženega prehoda od enega do drugega (angl. *vendor lock-in*) in

– potreben je dostop do ponudnika oziroma zagotovljen mora biti dostop do omrežja. Prav tako se ponudniki lahko nahajajo na oddaljeni lokaciji, kar poveča odzivni čas.

2.2 Namestitveni model

Oblak je lahko nameščen na različne načine, ti pa so odvisni od zahtev in potreb organizacij. Namestitveni model (angl. *deployment model*), kot ga definira NIST, opisuje namen uporabe oblaka in način namestitve [1,3,16]. Model vključuje javni, zasebni, skupnostni in hibridni oblak.

Javni oblak

Javni oblak je oblak, ki je javno dostopen komurkoli in je dostopen preko javnega omrežja – spleta. Ponudniki teh oblakov priskrbijo strojne in programske vire, ki se delijo med vsemi uporabniki oblaka. Uporaba virov javnih oblakov obsega storitve, ki jih ponudnik zaračuna uporabnikom. Javni oblaki predstavljajo največje tveganje glede varnosti, saj so viri dani v souporabo vsem njihovim uporabnikom. Posledično morajo uporabniki zaupati ponudniku in po potrebi vpeljati lastno zaščito podatkov [2, 3, 16, 20].
Najbolj razširjeni ponudniki javnih oblakov so Amazon[1], Digital Ocean[2], Google [3] in Microsoft [4].

Zasebni oblak

Zasebni oblaki so po svojem delovanju podobni javnim, razlika je le ta, da so vsi viri oblaka namenjeni enemu podjetju oziroma organizaciji. Oblak upravlja ta organizacija ali pa tretja oseba znotraj prostorov organizacije ali pa znotraj

[1]Amazon Web Services – `https://aws.amazon.com`
[2]Digital Ocean – `https://www.digitalocean.com/`
[3]Google Compute Engine – `https://cloud.google.com/`
[4]Microsoft Azure – `https://azure.microsoft.com/`

prostorov tretje osebe. V primerjavi z javnimi oblaki so v večini primerov strojne zmogljivosti bistveno manjše. Zasebni oblaki so primerni za podjetja, ki že imajo v lasti strojno opremo ali pa želijo obdržati nadzor nad podatki, varnostno politiko in optimizacijo celotnega sistema. Slabost zasebnega oblaka je cena njegove postavitve, saj je potreben zakup večjih količin strojne opreme, zahtevnejša sta tudi administracija in upravljanje s sistemom [1, 3, 16, 20].
Nekateri izmed ponudnikov, ki ponujajo postavitev ali pa sisteme zasebnih oblakov, so VMWare [5], OpenStack [6], Amazon [7], Microsoft in Rackspace [8].

Skupnostni oblak

Kadar ima več organizacij skupne zahteve in potrebe, kot sta varnost in zasebnost, lahko uporabijo skupnostni oblak. Ta jim omogoča združevanje in deljenje računskih virov, podatkov in zmogljivosti, kar organizacijam zniža stroške poslovanja. Skupnostni oblaki so podobni zasebnim oblakom s to razliko, da dostopa do sistema nima le ena organizacija, ampak več [1, 3, 16].

Hibridni oblak

Hibridni oblak omogoča povezovanje javnih, zasebnih in skupnostnih oblakov in s tem delovanje oblaka kot celote, neodvisno od tega, pri katerem ponudniku se storitev izvaja. S tem organizacije niso omejene na zgolj enega ponudnika oblačnih storitev. Ena od mogočih uporab hibridnih oblakov je tudi ta, da v primeru velikega povpraševanja po storitvah ali ob preveliki obremenitvi sistema organizacije uporabijo le določene storitve javnih oblakov [1, 3, 16].

[5] VMWare private cloud – `https://www.vmware.com/cloud-computing/private-cloud`

[6] Open stack private cloud – `https://www.openstack.org/marketplace/hosted-private-clouds/`

[7] Amazon private cloud – `https://aws.amazon.com/vpc/`

[8] Rackspace private cloud – `https://www.rackspace.com/cloud/private`

2.3 Storitveni model

Storitveni model (angl. *sevice model*) po opredelitvi NIST zajema razpon različnih tipov storitev, ki jih nudijo oblačni sistemi. Predstavljen je kot sklad, ki ga sestavljajo trije sloji, vsak sloj pa je odvisen od predhodnega. Model opredeli odgovornosti tako ponudnika kot uporabnika storitve znotraj vsakega sloja [1,3,16].

Najnižji sloj predstavlja **infrastruktura kot storitev – IaaS**, ki omogoča dodeljevanje osnovnih računskih virov, to so procesna moč, pomnilnik in omrežje. Te vire uporabnik izkorišča za nameščanje in izvajanje poljubne programske opreme, kar lahko vključuje operacijski sistem in aplikacije. Uporabnik ne upravlja ali nadzoruje infrastrukture oblaka, ampak ima nadzor zgolj nad nameščeno programsko opremo (operacijski sistem in aplikacije), shrambo in v določenih primerih nastavitvami omrežnih komponent (npr. požarna pregrada) [16].
Srednji sloj predstavlja **platforma kot storitev – PaaS**, ki omogoča postavitev aplikacij na infrastrukturo oblaka. Te aplikacije morajo biti razvite s pomočjo programskih knjižnic, storitev in programskih jezikov, podprtih s strani ponudnika oblaka. Uporabnik ne upravlja ali nadzoruje infrastrukture oblaka, ki vključuje strežnike, shrambo in operacijski sistem, ampak ima nadzor zgolj nad postavljenimi aplikacijami in njihovimi nastavitvami za spreminjanje konfiguracije njihovega gostovanja [16].
Najvišjo raven storitvenega modela predstavlja **programska oprema kot storitev – SaaS**, ki omogoča uporabo aplikacij ponudnika, ki se izvajajo na infrastrukturi oblaka. Aplikacije so dostopne preko različnih odjemalcev, bodisi preko spletnega brskalnika bodisi preko programskega vmesnika API. Uporabnik ne upravlja ali nadzoruje infrastrukture oblaka, ki vključuje strežnike, shrambo, operacijski sistem in aplikacije, ampak ima nadzor zgolj nad določenimi nastavitvami aplikacij ponudnika [16].

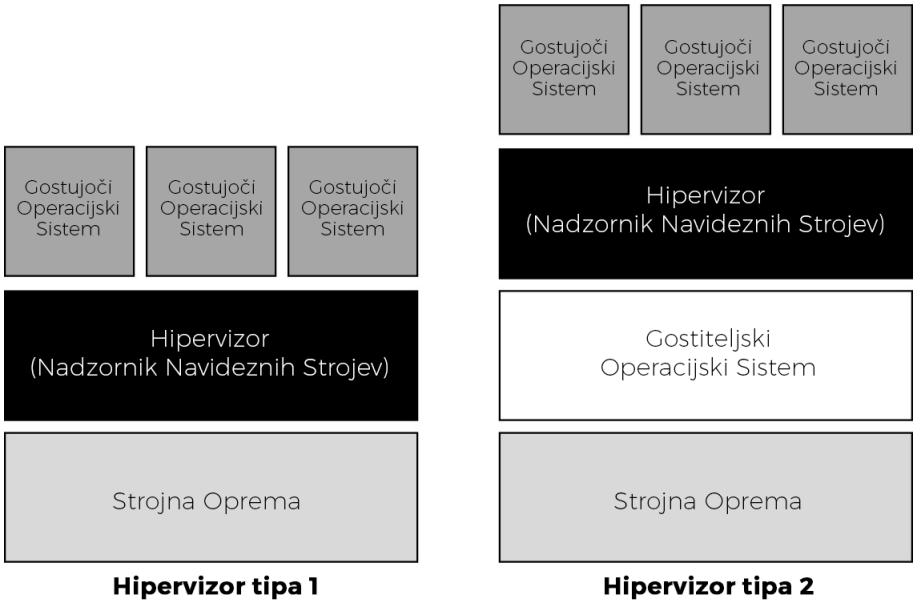
Poleg zgoraj navedenih modelov se pojavlja vedno več modelov, ki rešujejo specifične probleme razvijalcev. Ti vključujejo podatkovno bazo kot storitev – DBaaS (angl. *database as a service*), shranjevanje kot storitev – STaaS (angl. *storage as a service*), varnost kot storitev – SECaaS (angl. *security as a service*), namizje kot storitev – DaaS (angl. *sesktop as a service*) in sisteme za podporo

storitev – BaaS [9, 21].

2.3.1 Infrastruktura kot storitev

Večina storitev ponudnikov sloja IaaS temelji na virtualizaciji. Virtualizacija je temelj tehnologije računalništva v oblaku, ki omogoča, da se na enem fizičnem stroju sočasno izvaja več navideznih strojev. Omogoča postavitev navideznih strojev (angl. *virtual machine*), ki so vsebovalniki (angl. *container*), ki jim dodelimo različne strojne vire in operacijski sistem [3, 18, 22].
Za delovanje virtualizacije je potreben nadzornik navideznih strojev – VMM (angl. *virtual machine manager*) oziroma hipervizor (angl. *hypervisor*), ki omogoča deljenje fizičnih virov med navideznimi stroji. Hipervizor vsakemu operacijskemu sistemu dodeli lastne navidezne vire, ki posnemajo delovanje fizičnih virov, kot so centralna procesna enota, pomnilnik, shramba, omrežja in ostali strojni viri, hkrati pa nadzoruje porabo teh virov in zagotavlja, da vsi operacijski sistemi delujejo nemoteno drug od drugega.

Hipervizorje delimo glede na njihov način izvajanja, in sicer na hipervizorje tipa 1 in tipa 2. Njihova struktura je prikazana na sliki 2.1. Operacijski sistem, ki je nameščen na navideznem stroju, imenujemo gostujoči operacijski sistem [3].



Slika 2.1: Struktura hipervizorjev tipa 1 in tipa 2

Hipervizorji tipa 1 se izvajajo neposredno na fizični strojni opremi in za svoje delovanje ne potrebujejo operacijskega sistema. Ti hipervizorji omogočajo popolno virtualizacijo, saj popolnoma simulirajo strojno opremo, na kateri delujejo.

Hipervizorji, ki so nameščeni na operacijskih sistemih in ne neposredno na strojni opremi, so tipa 2. Aplikacije, ki delujejo znotraj navideznih strojev tipa 2, so običajno počasnejše od aplikacij, ki delujejo na hipervizorjih tipa 1, vendar so zaradi svoje prenosljivosti zelo priljubljene.
Virtualizacija odpravi potrebo po več fizičnih strojih, ko programska oprema zahteva različne operacijske sisteme. V primeru povišane obremenitve lahko hitro postavimo nove navidezne stroje, v primeru napak pa repliciramo njihove varnostne kopije. Razvijalci lahko s pomočjo virtualizacije oblikujejo lastne sisteme, ki so prilagojeni njihovim zahtevam in jim omogočajo vzporedno in porazdeljeno izvajanje programov in nalog. Najbolj razširjeni ponudniki storitev IaaS so Amazonov EC2, Microsoft Azure, Google Compute Engine in Digital Ocean [1–3, 9].

2.3.2 Platforma kot storitev

Za razvoj spletnih aplikacij ali storitev z visoko zanesljivostjo delovanja sta potrebni strojna in programska oprema, kar pogosto vključuje podatkovno bazo, spletni strežnik, porazdeljevanje obremenitve in druge podporne storitve. Vso opremo je treba nadzorovati in nadgrajevati. Virtualizacija omogoča hitro postavitev navideznih strojev, vendar je upravljanje in nameščanje vse potrebne programske opreme zamudno [1–3].
Platforma kot storitev (PaaS) nudi programsko okolje za razvoj in testiranje programske opreme, platforma pa poskrbi za distribucijo aplikacije čez oblačno infrastrukturo, brez poznavanja njene lokacije in delovanja. Cilj sistemov PaaS je avtomatizacija razvoja in postavitve visoko dostopnih aplikacij in posledično zmanjšanje izdatkov za nakup strojne in programske opreme ter stroškov konfiguracije in upravljanja s to opremo. Večina ponudnikov se zanaša na sloj IaaS za oskrbo s strojno opremo, postavitev in konfiguracijo navideznih strojev in/ali strežnikov ter nameščanje in nadgrajevanje vse potrebne programske opreme. Aplikacije, razvite na sistemih PaaS, spadajo v storitveni model SaaS [2, 3, 22].

Obstajajo različne vrste sistemov PaaS, od polno delujočih razvijalskih platform, kot so Windows Azure, Red Hat OpenShift, Heroku, Google App Engine, do modularnih razvijalskih ogrodij, kot sta Drupal in Squarespace, kjer z uporabo že razvitih modulov pridemo do karakteristik storitev PaaS [3].

2.3.3 Programska oprema kot storitev

Glavna značilnost aplikacij, ki se nahajajo znotraj storitvenega modela SaaS, je, da za njihovo uporabo ni potrebna namenska strojna in/ali programska oprema, saj se aplikacije izvajajo na oddaljenih strežnikih in so na voljo na zahtevo. Dostop do aplikacije je mogoč preko interneta, v večini primerov preko spletnega brskalnika [1–3, 22]. Primeri nekaterih takih aplikacij so Googlov poštni odjemalec Gmail in koledar Calendar, družbeno omrežje Facebook in CRM platforma Salesforce.

V splošnem so vsem aplikacijam SaaS skupne naslednje lastnosti [3]:

- **dostopnost:** do aplikacij odjemalci dostopajo na zahtevo, preko spletnega brskalnika in spleta. Za uporabo aplikacije ne potrebujejo posebne strojne ali programske opreme,
- **plačilo po porabi:** običajno se storitev izvaja preko naročnine ali sistema plačila po porabi,
- **vzdrževanje:** ponudnik storitve skrbi za vzdrževanje storitve ne glede na lokacijo izvajanja različnih programskih komponent,
- **visoka dostopnost:** ponudnik poskrbi za porazdelitev aplikacije preko več navideznih strojev in omogoča prilagajanje glede na obremenitev,
- **nizek finančni vložek:** zaradi zmanjšane distribucije, vzdrževanja in minimalnih stroškov za končnega uporabnika so aplikacije SaaS cenovno ugodnejše kot klasične aplikacije,
- **večnajemniška arhitektura:** vsi uporabniki aplikacije si delijo vire infrastrukture in
- **ena različica:** vsi uporabniki uporabljajo isto različico aplikacije, tako da so te skladne.

2.3.4 Podporni sistem kot storitev

Podporni sistemi kot storitev (BaaS) razvijalcem omogočajo integracijo različnih storitev oblaka v mobilne in spletne aplikacije. Večinoma te storitve vsebujejo različne načine shranjevanja podatkov, pošiljanje potisnih sporočil (angl. *push notifications*), avtorizacijo in avtentikacijo uporabnikov, vodenje različnih sej in integracijo z družbenimi omrežji. Z uporabo sistemov BaaS razvijalci skrajšajo razvojni čas, saj jim omogočajo enoten način za upravljanje z oblačnimi storitvami in njihovimi podatki [8–12].

Za uporabo sistema BaaS je potrebna intergracija razvojnega orodja SDK ali ustreznih knjižnic v aplikacijo [8, 12]. Ta preko programskih vmesnikov API komunicira s sistemom BaaS in omogoča izvajanje različnih operacij storitev. Storitve registriramo v sistem in jih aplikacijam omogočimo v uporabo.

Poglavje 3

Zasnova sistema BaaS

3.1 Motivacija

Za razvoj spletne ali mobilne aplikacije, ki uporabnikom omogoča shranjevanje datotek, mora razvijalec priskrbeti strežnike, relacijsko ali nerelacijsko podatkovno bazo, dovolj velike kapacitete diskov za shranjevanje datotek ter sistem za avtentikacijo in avtorizacijo uporabnikov. Vso navedeno programsko opremo je treba najprej namestiti in skonfigurirati, nato pa še integrirati v aplikacijo. Po namestitvi se razvijalci lahko osredotočijo na izdelavo logike aplikacije.
Celoten proces postavitve je zamuden, zlasti, ko je treba v aplikacijo dodati novo funkcionalnost, ki zahteva dodatno programsko opremo. Z uporabo storitev oblaka se namestitvi in konfiguraciji strojne in/ali programske opreme izognemo, saj to opravi ponudnik storitve. Kljub temu pa je za uporabo vsake storitve treba v aplikacijo integrirati ponudnikove programske vmesnike (API) in/ali razvojna orodja (SDK) [8, 9, 23].

Uporaba sistemov BaaS odpravlja težave s postavitvijo in upravljanjem strežnikov in/ali storitev. To skrajša čas, potreben za izdajo aplikacije (angl. *time-to-market*), in posledično tudi stroške. Aplikacija je odvisna zgolj od ene same programske knjižnice, komunikacija s storitvami pa poteka preko enotnega vmesnika [8, 15, 23]. Razvijalci znotraj sistema BaaS pridobijo t. i. prostor, v katerega aplikacija, preko storitev, shranjuje podatke, potrebne za svoje delovanje [8, 11]. Sistemi nudijo nabor že pripravljenih storitev, med katere spadajo shranjevanje

datotek, relacijskih podatkov, avtentikacija, avtorizacija in registracija uporabnikov ter delo z družbenimi omrežji. S storitvami upravljamo preko administrativne aplikacije, za njihovo uporabo pa je potrebna vključitev razvojnega orodja SDK oziroma programskega vmesnika API v aplikacijo [8–12]. Če želimo v aplikacijo dodati novo storitev, lahko uporabimo katero izmed že obstoječih storitev ponudnika ali pa sistem nadgradimo z lastno implementacijo storitve, če ponudnik to omogoča [8, 9, 23]. Prostori so med seboj ločeni, kar omogoča večnajemniško arhitekturo. S pomočjo različnih mehanizmov lahko določamo pravice za dostop do posamezne storitve. Tak primer so ACL (angl. *access control list*) dovoljenja, ki definirajo pravice za branje in pisanje posameznega uporabnika [8, 15].
Uporaba sistemov BaaS je razširjena zlasti med razvijalci mobilnih aplikacij, saj za razvoj svojih idej ne potrebujejo znanja iz postavitve strežnikov in razvoja strežniških aplikacij.

Za dostop do sistema BaaS aplikacije potrebujejo t. i. dostopne ključe (angl. *access keys*) [8]. Na tržišču je veliko aplikacij, ki so zaradi slabe ozaveščenosti in neprimernega upravljanja z dostopnimi ključi ranljive. Ranljive so zato, ker lahko iz njih, s pomočjo analize in orodij, pridobimo njihove ključe, s pomočjo katerih lahko razvijemo zlonamerno aplikacijo, ki ima enaka dovoljenja kot prvotna aplikacija [14, 15].
Da je kaj takega mogoče izvesti, sta dokazali skupini raziskovalcev, ki sta razvili dva ločena sistema, PlayDrone in HAVOC [14, 15]. PlayDrone je analiziral aplikacije, objavljene na Google Play, in ugotovil, da je mogoče iz veliko aplikacij pridobiti dostopne ključe do storitve Amazon Web Services (AWS) in Facebooka, ki jih razvijalci uporabljajo v svojih aplikacijah. Sistem HAVOC pa je bil osredotočen na sisteme BaaS in je analiziral aplikacije Android in iOS, iz katerih je pridobil dostopne ključe do ponudnikov sistemov BaaS, kar jim je omogočilo dostop do podatkov več tisoč aplikacij.

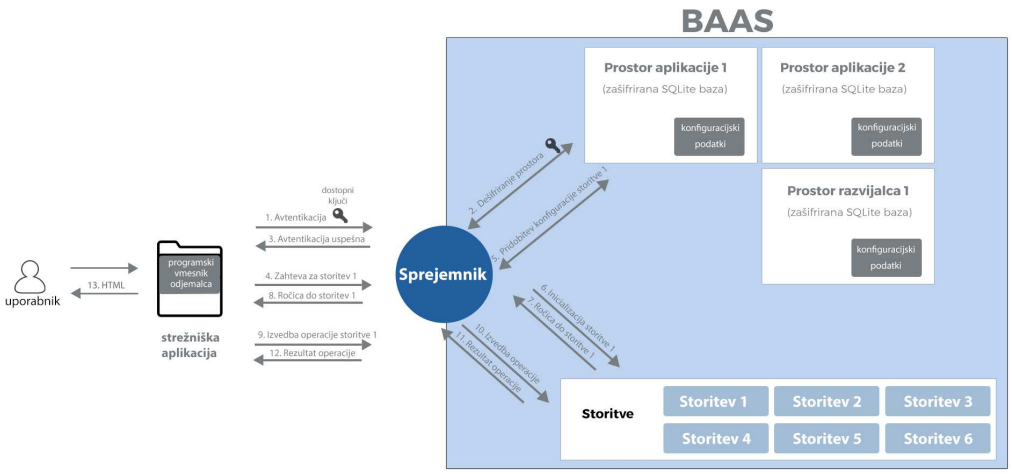
Za odpravo pomanjkljivosti, ki so bile izpostavljene v omenjenih raziskavah, bomo zasnovali in razvili nov sistem BaaS, katerega uporaba bo mogoča zgolj v strežniških aplikacijah. S takim načinom uporabe zamenjamo shranjevanje dostopnih ključev iz aplikacij, nad katerimi imajo uporabniki popoln nadzor, na strežnike, ki jih nadzirajo razvijalci. Taka zasnova od razvijalcev zahteva bistveno več znanja,

je pa neprimerljivo varnejša, saj ne prihaja do izpostavljenosti dostopnih ključev. Za uporabo sistema v mobilnih aplikacijah bo potrebna izdelava posredniških programskih vmesnikov, preko katerih bodo aplikacije komunicirale s sistemom na varen način. Za doseg čim višje stopnje varnosti bodo vsi podatki entitet med sabo izolirani, to pa bomo dosegli s pomočjo različnih kriptografskih algoritmov.

3.2 Zasnova sistema

Uporaba sistema se v večini primerov začne z interakcijo med aplikacijo in uporabnikom. Če se aplikacija za uporabo različnih storitev nanaša na sistem BaaS, potrebuje mehanizme, preko katerih komunicira s sistemom. Za komunikacijo v zasnovanem sistemu skrbi programska knjižnica, ki predstavlja komponento odjemalca in omogoča izmenjavo sporočil med aplikacijo in sistemom po vnaprej definiranem protokolu. Komponenta, s katero programski vmesnik odjemalca komunicira in predstavlja vstopno točno sistema, je sprejemnik (angl. *receiver*). Ta sprejema povezave odjemalcev, jih avtorizira in avtenticira ter jim nudi uporabo različnih storitev. Vsaka storitev je dostopna zgolj preko sprejemnika in odjemalcem omogoča izvajanje različnih operacij. Za izvajanje storitev je potreben skupen programski vmesnik, s pomočjo katerega sprejemnik dosega izvajanje različnih operacij storitve.

Glavni cilj sistema je shranjevanje različnih oblik podatkov aplikacij. V ta namen vsaka aplikacija v sistemu pridobi poseben prostor za shranjevanje podatkov, do katerega dostopa preko programske knjižnice odjemalca. Dostop do tega prostora je mogoč le preko sprejemnika, njegovi podatki pa so šifrirani zaradi izolacije in zaščite podatkov. Za delovanje vseh predstavljenih komponent je treba shranjevati različne oblike podatkov sistema. V ta namen sprejemnik vsebuje komponento, ki mu omogoča shranjevanje vseh podatkov, potrebnih za delovanje sistema. Povezava vseh komponent je prikazana na sliki 3.1.



Slika 3.1: Diagram, ki prikazuje komunikacijo med komponentami

Za realizacijo zgoraj opisanega sistema postavimo naslednje zahteve:

- dostop do sistema je omogočen zgolj strežniškim aplikacijam,
- uporaba sistema v nestrežniških aplikacijah poteka preko t. i. posredniških programskih vmesnikov, ki razkrivajo zgolj funkcionalnost strežniške aplikacije;
- vsi podatki sistema so šifrirani s pomočjo kriptografskih algoritmov;
- sistem mora omogočati sočasno uporabo več odjemalcem, za kar je potrebna večnajemniška arhitektura;
- vsi podatki aplikacij in razvijalcev so shranjeni v prostorih;
- vsaka aplikacija in razvijalec imata lasten prostor, ki je izoliran od ostalih prostorov;
- nepooblaščen dostop do enega prostora ne sme pripeljati do razkritja podatkov drugih prostorov;
- sistem mora vsebovati prijavni sistem za avtentikacijo in avtorizacijo entitet;

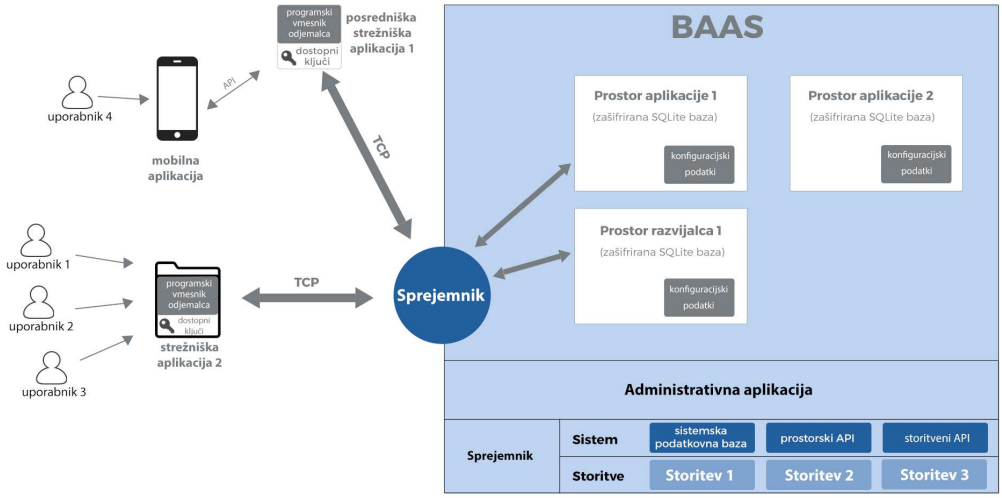
- vsaka aplikacija mora za dostop do sistema dokazati svojo identiteto;
- sistem mora omogočati uporabo različnih storitev sistema;
- komunikacija med aplikacijo in sistemom poteka preko sprejemnikov in
- razvijalci vse nastavitve aplikacij upravljajo preko administrativne aplikacije.

3.3 Komponente

Na osnovi navedenih zahtev sistema smo zasnovali nov sistem, ki ga sestavljajo naslednje komponente:

- sprejemnik,
- administrativna aplikacija,
- storitveni programski vmesnik API,
- sistemska podatkovna baza,
- prostori in prostorski programski vmesnik API in
- programski vmesnik odjemalca.

Komponente in njihova medsebojna povezava so predstavljene na sliki 3.2. V nadaljevanju bomo vse komponente in njihovo delovanje predstavili.



Slika 3.2: Struktura zasnovanega sistema BaaS

3.3.1 Prostor in sistemska podatkovna baza

Sistem mora ločevati med dvema entitetama – med razvijalci, ki ustvarjajo in upravljajo svoje aplikacije, ter aplikacijami, ki uporabljajo storitve sistema. Za shranjevanje podatkov aplikacij sistem vsaki entiteti dodeli svoj prostor. V njem so shranjeni vsi dostopni podatki in konfiguracije storitev aplikacije. Pridobitev nepooblaščenega dostopa do prostora lahko pripelje do razkritja njenih podatkov. Da sistem to prepreči, so vsi podatki prostora šifrirani, za dostopanje do prostora pa mora imeti entiteta poseben ključ, ki podatke odšifrira. Hkrati so prostori tudi prijavni sistem – če ima entiteta veljaven simetričen ključ do svojega prostora, s tem tudi dokaže svojo verodostojnost. Podatki znotraj prostora morajo biti pravilno strukturirani in morajo omogočati izvajanje poizvedb po njih, zato smo prostore realizirali s pomočjo podatkovne baze SQLite. Šifriranje omogoča, da lahko na enem strežniku hranimo več tovrstnih prostorov in tako dosežemo izolacijo. Če se zgodi nepooblaščen dostop do strežnika, vsiljivec ne more dešifrirati podatkov prostora v doglednem času niti ne more v primeru pridobitve dostopa do enega prostora dostopati do ostalih prostorov. Za večjo uporabnost prostorov lahko aplikacije določene podatke v njih tudi shranjujejo.

Dostop do sistema se razlikuje glede na entiteto, ki do sistema dostopa. Postopek prijave se med entitetami razlikuje, vendar gre v osnovi za enak proces – odšifriranje prostora. Razvijalci do sistema dostopajo preko administrativne aplikacije, ta od njih zahteva vnos uporabniškega imena in gesla, medtem ko aplikacije dostopajo preko sprejemnikov, ki od njih zahtevajo geslo za dostop do njihovega prostora. Poleg tega mora aplikacija imeti še dodatne asimetrične ključe, s pomočjo katerih dokaže svojo identiteto.

Poleg prostorov mora sistem shranjevati določene informacije, ki so namenjene za identifikacijo aplikacij, prostorov in storitev. V ta namen sistem vsebuje sistemsko podatkovno bazo MySQL.

Ta vsebuje:

- seznam vseh prostorov in
- seznam vseh storitev, ki jih je mogoče uporabljati.

3.3.2 Sprejemnik

Sprejemnik je vstopna točka sistema BaaS za aplikacije. Preko njega aplikacije dostopajo do storitev sistema. Deluje na načelu modela odjemalec-strežnik (angl. *client-server model*).

Njegove naloge so naslednje:

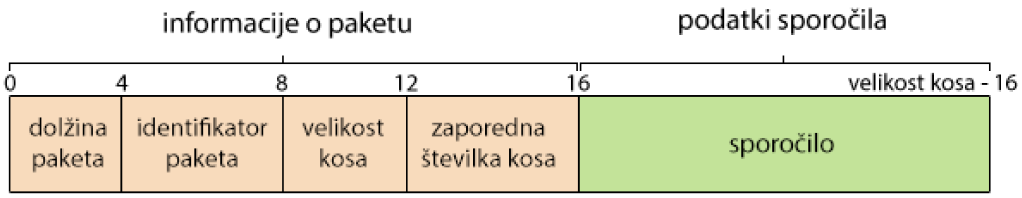
- sprejemanje povezav aplikacij,
- avtentikacija in avtorizacija aplikacij,
- inicializacija programskih knjižnic storitev in
- vodenje komunikacije med aplikacijo in posamezno storitvijo.

Za nemoteno sprejemanje povezav se sprejemnik izvaja v več nitih in uporablja odcepitev procesov (angl. *process forking*). Uporaba storitve vedno poteka preko sprejemnika, ki na določenih vratih (angl. *port*) TCP sprejema nove povezave, jih avtenticira in avtorizira s pomočjo sistemske podatkovne baze ter gesla za dostop

do prostora aplikacije. Če sta procesa avtentikacije in avtorizacije uspešna, sta aplikacija in sprejemnik uspešno povezana. To pomeni, da lahko aplikacija od sprejemnika zahteva katerokoli storitev, ki je definirana v njegovem prostoru.

Sporočila

Komunikacija med aplikacijo in sprejemnikom mora biti zanesljiva in kar se da hitra, zato poteka preko povezave TCP. Med seboj si izmenjujeta podatke v obliki paketov, ti pa vsebujejo sporočilo in dodatne informacije o stukturi. Ker so paketi lahko poljubno veliki, mora pošiljatelj paket razdeliti na več kosov, prejemnik mora to zaznati in kose paketa rekonstruirati v prvotno obliko. Podatki sporočila so strukturirani s pomočjo sintakse JSON. Sestava paketa je predstavljena na sliki 3.3.



Slika 3.3: Sestava paketa

Uporaba storitev

Sprejemnik aplikacijam omogoča izvajanje oddaljenih metod storitev (angl. *remote method invocation*). Ko aplikacija potrebuje določeno storitev, sprejemniku posreduje sporočilo, ki vsebuje identifikator zahtevane storitve. Sprejemnik najprej preveri, ali ta identifikator obstaja v sistemski podatkovni bazi, nato pa iz prostora aplikacije prebere konfiguracijo storitve, ki vsebuje vse dostopne podatke. Če je preverjanje uspešno, ustvari instanco programskega razreda, ki predstavlja zahtevano storitev in vsebuje vse mehanizme ter protokole za uporabo storitve znotraj sprejemnika. Dokler je povezava med aplikacijo in sprejemnikom vzpostavljena,

sprejemnik v pomnilniku hrani to instanco, aplikaciji pa posreduje identifikacijsko številko, ki enolično identificira ročico storitve (angl. *service handle*) oziroma instanco storitve. Za izvedbo operacije storitve aplikacija sprejemniku posreduje ime operacije, argumente operacije in identifikacijsko številko ročice do storitve. Sprejemnik preveri, ali ta ročica obstaja, njeni instanci pa posreduje zahtevek za operacijo in njene argumente. Če je ta operacija uspešna, sprejemnik aplikaciji odgovori s sporočilom, ki vsebuje odgovor storitve, v primeru napake pa mu posreduje sporočilo z opisom napake. Ko aplikacija zaključi uporabo storitve, posreduje sprejemniku posebno sporočilo, ki odstrani ročico do storitve in instanco te storitve. Za delovanje zgoraj opisanega mora razred, ki predstavlja storitev, implementirati poseben programski vmesnik, ki vsebuje vse metode za tovrstno komunikacijo. Ta vmesnik vsebuje tudi metode, ki omogočajo nameščanje in odstranjevanje storitve ter njenih podatkov v prostore.

Prijava aplikacij

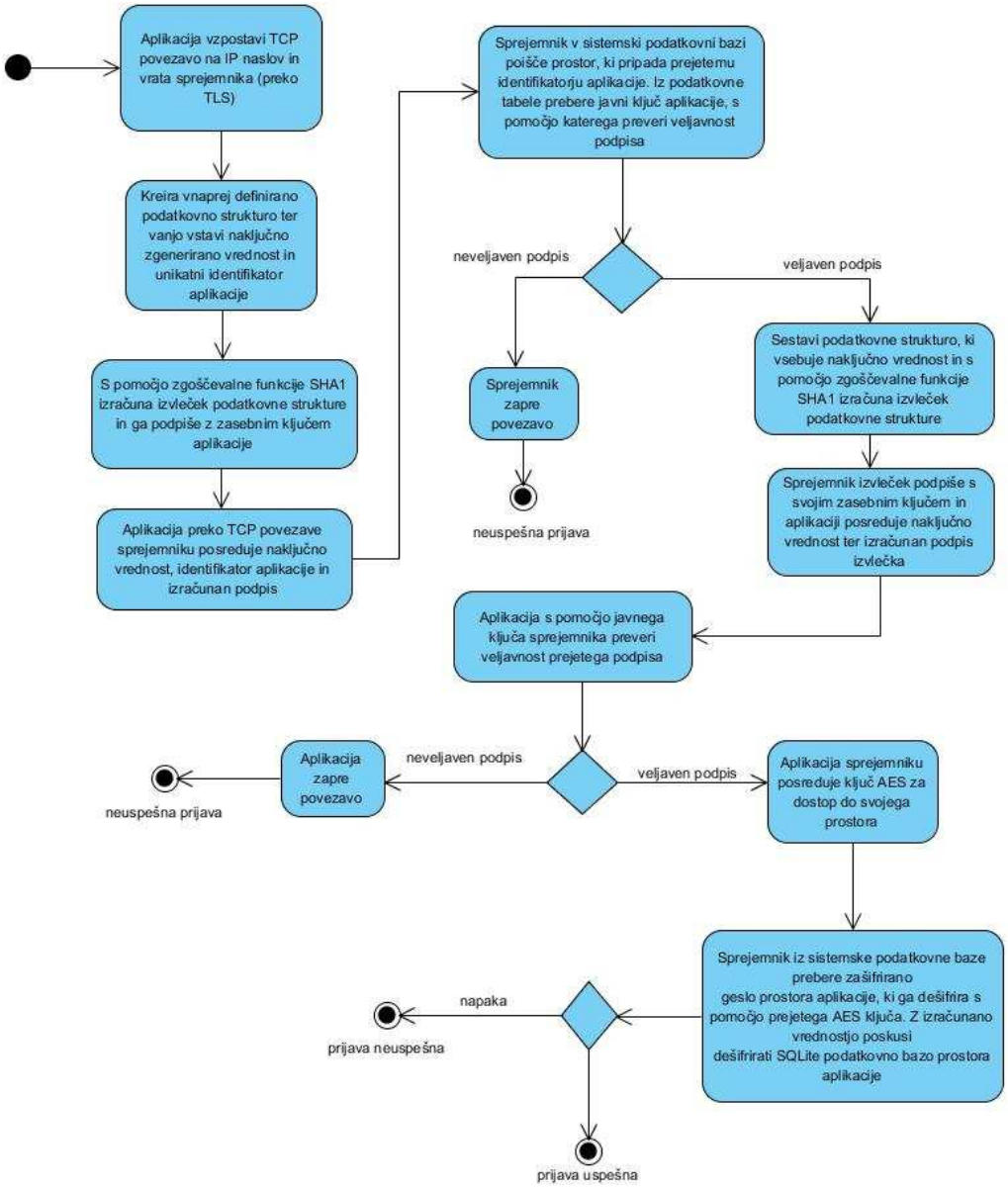
Komunikacija med aplikacijo in sprejemnikom poteka preko javnega omrežja, kar zahteva medsebojno dokazovanje verodostojnosti med aplikacijo in sprejemnikom.

Aplikacije potrebujejo za prijavo v sistem naslednje:

- unikaten identifikator aplikacije,
- javni in zasebni ključ aplikacije, ustvarjen z algoritmom RSA (velikosti 4096 bitov),
- javni ključ sprejemnika (velikost 4096 bitov) in
- ključ AES za dostop do prostora aplikacije (velikost 512 bitov).

Vse ključe in podatke aplikacija pridobi ob registraciji v sistem preko administrativne aplikacije.
Potek prijave je prikazan na sliki 3.4.

Login.jpg



Slika 3.4: Diagram poteka prijave aplikacije v sistem

3.3.3 Administrativna aplikacija

Administrativna aplikacija je namenska aplikacija, dostopna preko spletnega brskalnika, in se izvaja na istem strežniku kot sprejemnik. Njen cilj je s pomočjo

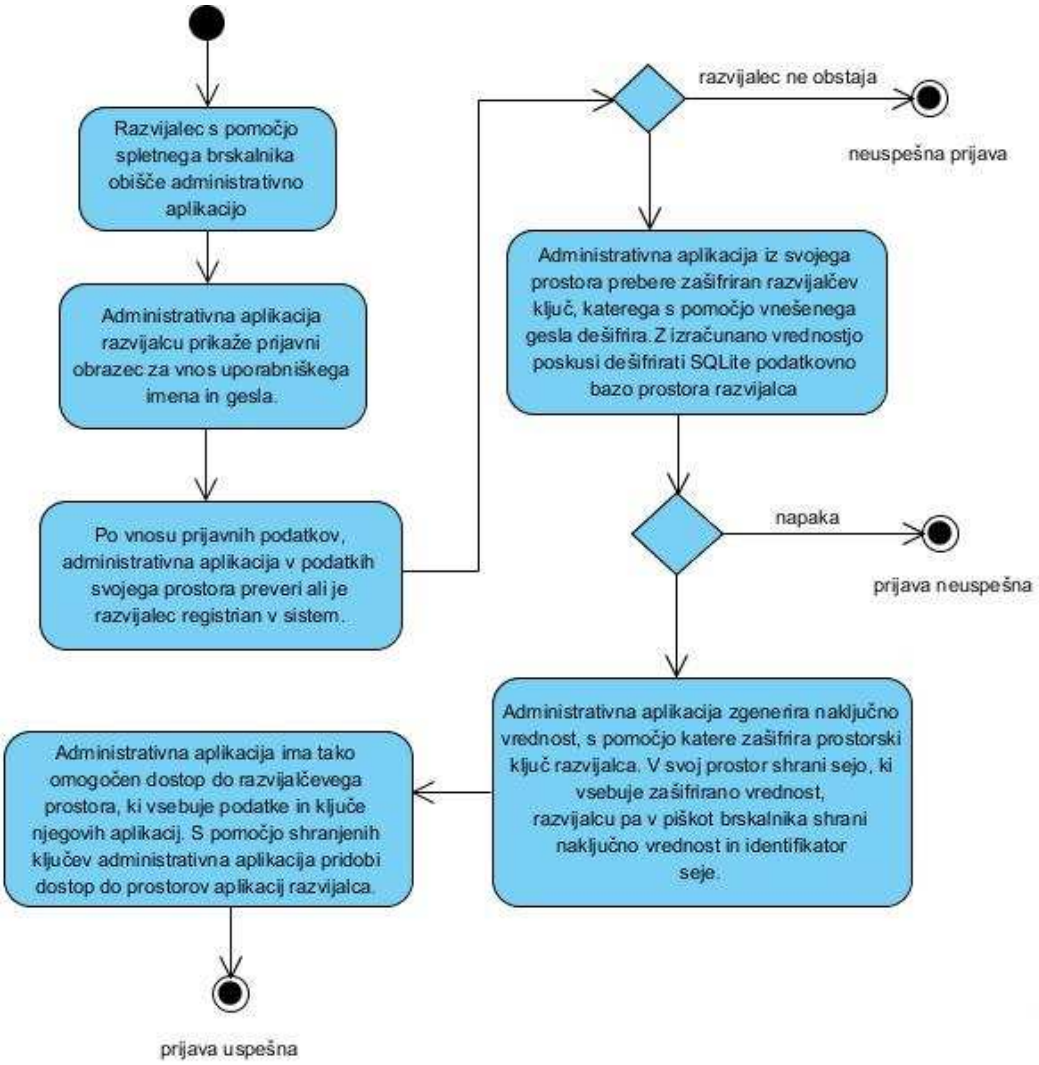
vizualnih gradnikov omogočiti naslednje:

- registracijo novih razvijalcev,
- avtentikacijo in avtorizacijo razvijalcev do svojih prostorov in prostorov njihovih aplikacij,
- ustvarjanje novih aplikacij,
- upravljanje podatkov in nastavitev prostora in
- upravljanje konfiguracij storitev.

Vsak prostor v sistemu je zaščiten s svojim šifrirnim ključem, ki pa ga administrativna aplikacija nima. V ta namen vsak razvijalec, tako kot aplikacija, pridobi svoj prostor, dostop do njega pa je mogoč s pravilnim uporabniškim imenom in geslom. Ob oblikovanju nove aplikacije se razvijalcu v njegov prostor shranijo dostopni ključi, ki so potrebni za dešifriranje prostora aplikacije. S pomočjo teh ključev administrativna aplikacija omogoča upravljanje konfiguracij aplikacij.
Tako kot vsaka aplikacija v sistemu tudi administrativna aplikacija pridobi svoj prostor, znotraj katerega shranjuje vse podatke o razvijalcih, ti pa so že v osnovi šifrirani.

Prijava razvijalcev

Proces prijave razvijalcev je drugačen od procesa prijave aplikacije, saj je tu treba dostopati do več prostorov. Proces prijave razvijalca v sistem je prikazan na sliki 3.5.



Slika 3.5: Diagram poteka prijave razvijalca v administrativno aplikacijo

3.3.4 Programski vmesnik odjemalca

Za komunikacijo strežniških aplikacij s sistemom je potrebna programska knjižnica odjemalca, ki omogoča uporabo zasnovanega sistema BaaS.

Ta vključuje naslednje:

- komunikacijo s sprejemnikom preko povezave TCP,

- implementacijo protokola za izmenjavo sporočil,
- uporabo storitev sistema,
- programsko knjižnico za šifriranje, digitalno podpisovanje podatkov in preverjanje digitalnih podpisov ter
- mehanizem za shranjevanje vseh dostopnih podatkov in ključev aplikacije.

Za uporabo sistema v mobilnih aplikacijah oziroma nestrežniških aplikacijah je obvezna izdelava strežniške aplikacije. Njen namen je varno shranjevanje vseh dostopnih podatkov in ponujanje programskega vmesnika, s katerim mobilna aplikacija nato komunicira. Spletna aplikacija mora vse zahteve interpretirati, se povezati s sistemom BaaS, izvesti določena opravila in posredovati odgovor. Tako nestrežniškim aplikacijam ni treba shranjevati občutljivih podatkov in tudi nimajo na voljo izvajanja operacij storitev sistema, ki bi ga lahko zlonamerno izkoriščali.

Poglavje 4

Razvoj sistema BaaS

V tem poglavju je predstavljena realizacija predhodno definirane zasnove podpornega sistema. Najprej predstavimo potrebno programsko opremo za začetek razvoja, od operacijskega sistema do programskega okolja in vseh potrebnih storitev. Temu bosta sledila implementacija komponent sistema in opis njihove strukture.

Za razvoj in testiranje sistema potrebujemo:

1. virutalizacijo za postavitev in izvajanje več navideznih strojev,
2. operacijski sistem, nameščen na navidezne stroje,
3. spletni strežnik, ki procesira zahtevke HTTP,
4. programsko okolje in knjižnice, potrebne za razvoj spletnih aplikacij,
5. omrežno dostopnost za sprejemanje in pošiljanje podatkov preko povezave TCP in
6. večnitnost za izvajanje programov s pomočjo več niti.

Virtualizacijo smo dosegli s pomočjo programske opreme Citrix Xen Server [1], ki je hipervizor tipa 1. S pomočjo slednjega smo na enem fizičnem stroju izvajali več navideznih strojev, ti pa so imeli nameščen operacijski sistem Linux Ubuntu

[1] XenServer – `http://xenserver.org/`

Server [2]. Za izvajanje spletnih aplikacij smo nanj namestili spletni strežnik Apache [3] in programske knjižnice, potrebne za izdelavo spletnih aplikacij, napisanih v programskem jeziku PHP. PHP nam omogoča uporabo različnih knjižnic operacijskega sistema, kar vključuje tudi delo s povezavami TCP, izvajanje programske kode v nitih in odcepitev procesov. Te knjižnice smo uporabili za izdelavo programa, ki s pomočjo več niti in procesov deluje v ozadju (angl. *background daemon*). Ta program je osnova sprejemnika BaaS, ki skrbi za komunikacijo med aplikacijami in sistemom.

Razvoj sistema bomo začeli z opisom implementacije posameznih komponent sistema, ki temeljijo na tehnikah in metodah, definiranih v prejšnjem poglavju.

4.1 Jedrne knjižnice

Jedro sistema je predstavljeno s sklopom različnih podpornih programskih knjižnic, ki smo jih uporabili pri razvoju posameznih komponent sistema. Te knjižnice zajemajo:

- kriptografsko knjižnico za šifriranje in odšifriranje podatkov, izračun in preverjanje digitalnih podpisov ter za ustvarjanje simetričnih in asimetričnih kriptografskih ključev,
- knjižnico za delo z nitmi in procesi,
- knjižnico za vzpostavljanje in sprejemanje povezav TCP ter pošiljanje in branje podatkov iz povezav TCP,
- knjižnico za uporabo podatkovnih baz MySQL in SQLite in
- knjižnico za pisanje dnevnikov za razhroščevanje.

Glavno komponento sistema predstavlja sprejemnik. Ta je sestavljen iz dveh delov, iz poslušalca in upravitelja. Poslušalec s poslušanjem na vnaprej določenih vratih TCP zazna nove odjemalce in s pomočjo odcepitve procesa vsakemu nameni

[2] Ubuntu Server – `http://www.ubuntu.com/server`
[3] Apache Web Server – `https://httpd.apache.org/`

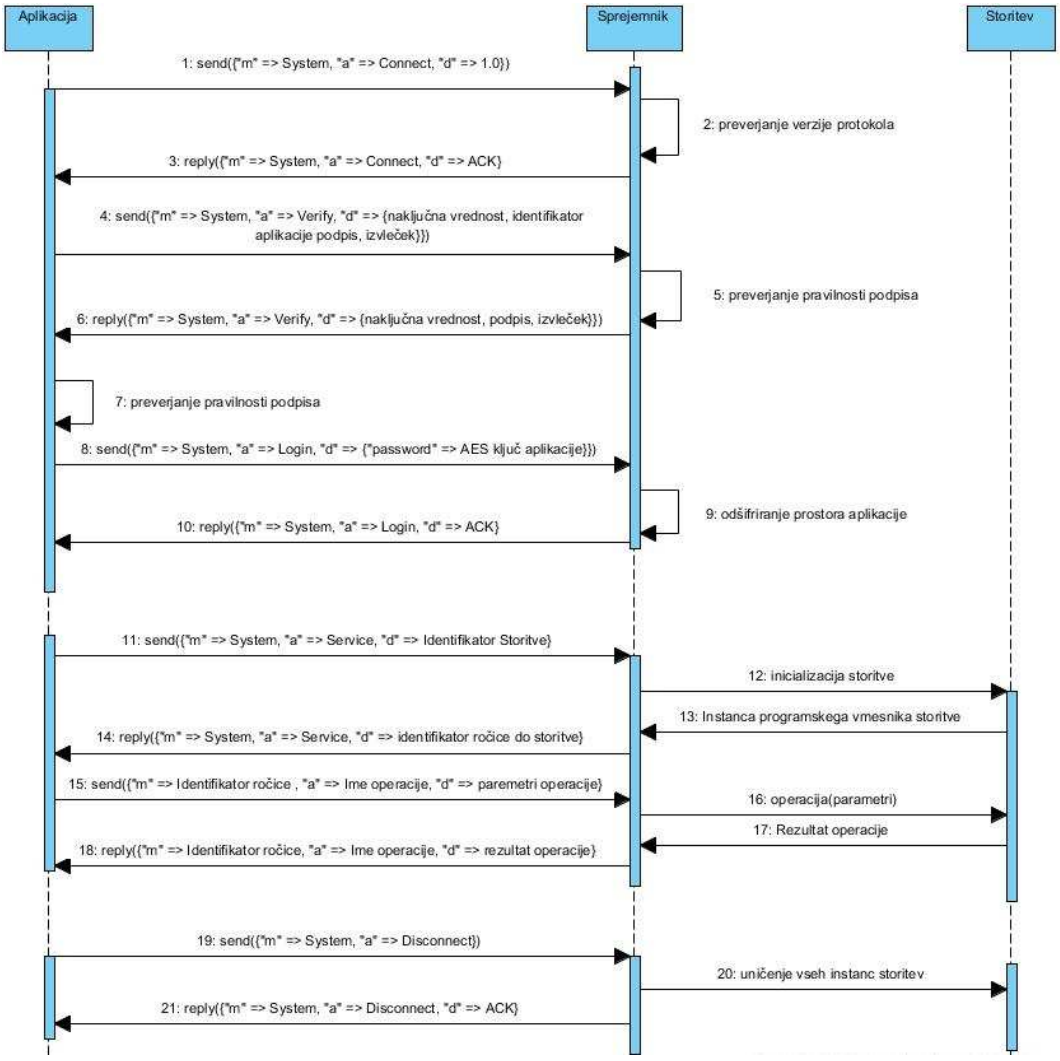
lasten proces. To omogoči nemoteno sprejemanje novih povezav brez zaviranja obstoječih. Odcepljen proces prevzame upravitelja, ki je odgovoren za izmenjavo sporočil med sistemom in aplikacijo. V programski kodi je predstavljen z jedrnim programskim razredom (angl. *core class*), ki združuje vse funkcionalnosti sistema. Uporabljata ga upravitelj in administrativna aplikacija, omogoča pa naslednje:

- izmenjavo sporočil z aplikacijo po vnaprej definiranem protokolu,
- izvajanje poizvedb SQL po sistemski podatkovni bazi,
- dešifriranje prostorov in izvajanje poizvedb SQL po njihovih podatkih,
- avtentikacijo in avtorizacijo aplikacij in
- logiko, potrebno za inicializacijo storitvenih vmesnikov in izvajanje njihovih operacij.

4.2 Protokol

Protokol za izmenjavo informacij med programskim vmesnikom odjemalca in sprejemnikom definira način branja, prejemanja in potrjevanja paketov ter stukturo izmenjanih sporočil. Vsa sporočila so predstavljena s paketi, ti pa so sestavljeni iz informacij o paketu in podatkov sporočila.
Izmenjava sporočil med pošiljateljem in prejemnikom poteka tako, da pošiljatelj najprej pošlje informacije o paketu, tem pa sledijo podatki sporočila. Prejemnik iz povezave bere, dokler ne prejme informacij o paketu – prvih 16 bajtov – s pomočjo katerih izračuna, koliko podatkov mora še prebrati, da prejme celotno sporočilo. Podatke o sporočilu shranjuje v začasen pomnilnik, ta proces pa ponavlja, dokler ne prejme vseh kosov paketa. Ko prejme vse podatke paketa, iz začasnega pomnilnika sestavi podatkovno strukturo sporočila JSON, ki jo nato interpretira. V primeru uspešno prejetega paketa pošiljatelju odgovori s potrditvenim sporočilom, ki vsebuje identifikacijsko številko prejetega paketa in posebno vrednost, ki označuje uspešno prejeti paket. V primeru napake ta vrednost označuje neuspešno prejeti paket, zato ga odjemalec ponovno pošlje. V primeru več zaporednih neuspešno poslanih paketov se povezava zapre.

Podatkovna struktura sporočila JSON vsebuje vnaprej definirane ključe, ki se uporabljajo za shranjevanje podatkov v komunikaciji. Potek komunikacije med aplikacijo in sprejemnikom je prikazan na sliki 4.1.



Slika 4.1: Diagram poteka komunikacije med programskim vmesnikom odjemalca in sprejemnikom

4.3 Podatkovne baze

Jedrni razred mora shranjevati in dostopati do podatkov o prostorih in storitvah sistema kot tudi do podatkov znotraj posameznega prostora. Zasnova sistema zahteva, da so vsi podatki posameznega prostora šifrirani. V ta namen smo uporabili programsko knjižico SQLCipher [4]. Ta razširja podatkovno bazo SQLite in ji dodaja transparentno šifriranje podatkov podatkovne baze s simetričnim ključem. Za uporabo prostorske in sistemske podatkovne baze smo razvili dva programska razreda, ki omogočata izvajanje naslednjih operacij nad podatkovnimi bazami MySQL in SQLite:

- ***query*** – izvajanje poizvedbe SQL,
- ***delete*** – brisanje podatkov iz podatkovne baze,
- ***update*** – posodabljanje podatkov v podatkovni bazi,
- ***insert*** – dodajanje novih podatkov v podatkovno bazo,
- ***row*** – pridobivanje podatkov o vrsticah rezultatov poizvedbe,
- ***count*** – pridobitev števila rezultatov poizvedbe in
- ***commit*** – izvajanje transakcij med podatkovnimi bazami.

Definirane operacije omogočajo iskanje ter upravljanje s podatki sistemske in prostorske podatkovne baze. Njihova struktura je odvisna od namena podatkovne baze, tega pa določa administrativna aplikacija.

Sistemsko podatkovno bazo sestavljajo naslednje podatkovne tabele:

Prostor – vsebuje podatke o vseh prostorih sistema in njihovih dostopnih ključih. Struktura podatkovne tabele je naslednja:

- ***id*** – identifikator prostora,
- ***encKey*** – šifriran ključ za dostop do prostorske podatkovne baze SQLite,

[4] SQLCipher – `https://www.zetetic.net/sqlcipher/`

- ***pubKey*** – javni ključ lastnika prostora,
- ***created*** – čas ustvarjanja prostora in
- ***lastAccess*** – čas zadnjega dostopa do prostora.

Storitev – vsebuje vse podatke o storitvah, ki so v sistemu registrirane in so vsem aplikacijam na voljo za uporabo. Struktura podatkovne tabele je naslednja:

- ***id*** – identifikator storitve,
- ***name*** – ime storitve,
- ***path*** – pot do programskega vmesnika storitve in
- ***config*** – struktura JSON, ki vsebuje začetno konfiguracijo storitve.

Podatkovno bazo prostora aplikacije sestavlja naslednja podatkovna tabela:

Storitev – vsebuje podatke o vseh storitvah, ki so aplikacijam omogočene v uporabo. Struktura podatkovne tabele je naslednja:

- ***id*** – identifikator storitve,
- ***serviceId*** – identifikator, ki povezuje storitev aplikacije s storitvijo v sistemski podatkovni bazi,
- ***name*** – unikatna vrednost, ki jo aplikacija uporablja za naslavljanje določene storitve, in
- ***config*** – struktura JSON, ki vsebuje konfiguracijo storitve.

Podatkovna baza prostora razvijalca je sestavljena iz naslednjih podatkovnih tabel:

Aplikacije – vsebuje podatke o vseh aplikacijah razvijalca. Struktura podatkovne tabele je naslednja:

- ***id*** – identifikator aplikacije,
- ***name*** – ime aplikacije,

- ***URL*** – URL-naslov aplikacije,
- ***spaceId*** – identifikator prostora v sistemski podatkovni bazi in
- ***encKey*** – šifriran ključ za dostop do prostorske podatkovne baze SQLite aplikacije.

Seje – vsebuje podatke o vseh sejah aplikacij razvijalca. Struktura podatkovne tabele je naslednja:

- ***id*** – identifikator seje,
- ***appId*** – identifikator aplikacije razvijalca,
- ***created*** – čas ustvarjanja seje,
- ***expires*** – čas poteka seje in
- ***encKey*** – šifriran ključ za dostop do prostorske podatkovne baze SQLite aplikacije.

4.4 Storitve

Za uporabo storitve v zasnovanem sistemu je potreben razvoj programskega razreda, ki omogoča izvajanje operacij storitve. Za inicializacijo razreda in vso komunikacijo med aplikacijo in storitvijo poskrbi jedrni programski razred. Storitev mora zgolj implementirati poseben programski vmesnik, ki definira vse potrebne metode, potrebne za integracijo v sistem. To vključuje nameščanje in odstranjevanje storitve v prostore, uporabo operacij storitev znotraj aplikacij in definicijo začetne konfiguracije storitve.

Definicija programskega vmesnika je naslednja:

```
class Service {
   protected $core;
   protected $name;
   protected $config;
```

```
    function __construct($c) {
     $this->core = $c;
    }
    function use($name, $config) {
     $this->name = $name;
     $this->config = $config;
    }
    abstract function install();
    abstract function uninstall();

    function handleOperation($operation, $args) {
     return call_user_func(array($this,$operation), $args);
    }
}
```

Z implementacijo zgoraj definiranega programskega vmesnika lahko dodamo poljubno storitev v sistem, potrebne so zgolj implementacija abstraktnih metod, definicija začetne konfiguracije in implementacija metod za izvajanje operacij storitve.

Vsako storitev mora razvijalec aplikacije omogočiti preko administrativne aplikacije in ji definirati ustrezno konfiguracijo. Ker vsaka storitev zahteva drugačno obliko nastavitev, ji sistem omogoča definicijo strukture konfiguracije. S pomočjo te strukture storitev definira pravila svoje konfiguracije, administrativna aplikacija pa z njeno pomočjo preverja pravilnosti vnosa. Podatki o tem so shranjeni v sistemski podatkovni bazi, v tabeli storitev, strukturirani pa so s pomočjo sintakse JSON. Vsak ključ predstavlja vrednost, ki jo mora razvijalec izpolniti, ta pa je predstavljena z objektom JSON, ki opisuje njen tip in njeno obveznost. Tipi podatkov ključev so lahko naslednji:

- niz znakov (angl. *string*),
- celo število (angl. *integer*),
- decimalno število (angl. *double*),

- seznam (angl. *array*) ali
- binarna vrednost (angl. *blob*).

Za doseg uporabnosti sistema je treba definirati nabor storitev, ki jih aplikacije lahko uporabljajo. S tem namenom smo implementirali naslednje storitve:

Relacijska podatkovna baza MySQL – omogoča uporabo lokalne ali oddaljene podatkovne baze. Njena začetna konfiguracija vsebuje naslov podatkovne baze, uporabniško ime, geslo in ime podatkovne baze. Operacije, ki smo jih podprli, omogočajo:

- ***query*** – izvajanje poizvedb SQL,
- ***delete*** – brisanje podatkov iz podatkovne baze,
- ***update*** – posodabljanje podatkov v podatkovni bazi,
- ***insert*** – dodajanje novih podatkov v podatkovno bazo,
- ***row*** – pridobivanje podatkov o vrsticah rezultatov poizvedbe,
- ***count*** – pridobitev števila rezultatov poizvedbe in
- ***select*** – izbiro podatkovne baze.

Shranjevanje datotek – omogoča shranjevanje datotek na strežnik sistema BaaS. Njena začetna konfiguracija vsebuje ime in pot mape, v katero se shranjujejo datoteke, in dostopne pravice, ki se dodelijo vsaki datoteki. Operacije, ki jih je mogoče izvajati nad to storitvijo, so naslednje:

- ***fopen*** – odpiranje datotek,
- ***fclose*** – zapiranje odprtih datotek,
- ***fwrite*** – pisanje v datoteke,
- ***fread*** – branje iz datoteke,
- ***chmod*** – spreminjanje pravic datotek in map,

- ***delete*** – brisanje datotek in map,
- ***mkdir*** – ustvarjanje map in
- ***fileExists*** – preverjanje, ali datoteka ali mapa obstaja.

Storitev za upravljanje uporabnikov – omogoča preprosto integracijo prijavnega sistema aplikacije. Nudi avtentikacijo, avtorizacijo in registracijo uporabnikov. Njena začetna konfiguracija vsebuje ime storitve, ki se uporablja za shranjevanje podatkov, in ime podatkovnih tabel, v katere se shranjujejo podatki o uporabnikih in sejah. Operacije, ki se izvajajo nad to storitvijo, so naslednje:

- ***login*** – prijava uporabnika z uporabniškim imenom in geslom,
- ***register*** – registracija novega uporabnika,
- ***getUsers*** – iskanje in pridobivanje podatkov o vseh registriranih uporabnikih,
- ***updateUser*** – posodabljanje podatkov registriranih uporabnikov,
- ***deleteUser*** – brisanje registriranih uporabnikov,
- ***logout*** – odjava uporabnika in
- ***validateSession*** – preverjanje veljavnosti seje uporabnika.

4.5 Administrativna aplikacija

Administrativno aplikacijo smo izdelali s pomočjo spletnih tehnologij, ki vključujejo HTML, CSS in JavaScript. Aplikacija je vstopna točka za razvijalce, ki jim s pomočjo vizualnih gradnikov omogoča upravljanje prostorov aplikacij. Poleg dostopanja in upravljanja prostorov mora skrbeti tudi za registracijo novih razvijalcev in aplikacij. Postopek registracije se razlikuje glede na entiteto, vendar gre v osnovi za enak postopek – tvorjenje prostora – razlika je le v strukturi podatkovne baze prostora. Za lažjo implementacijo registracije ima administrativna aplikacija vnaprej pripravljene predloge struktur podatkovnih baz za posamezno entiteto.

V primeru registracije novega razvijalca v sistem je potek naslednji:

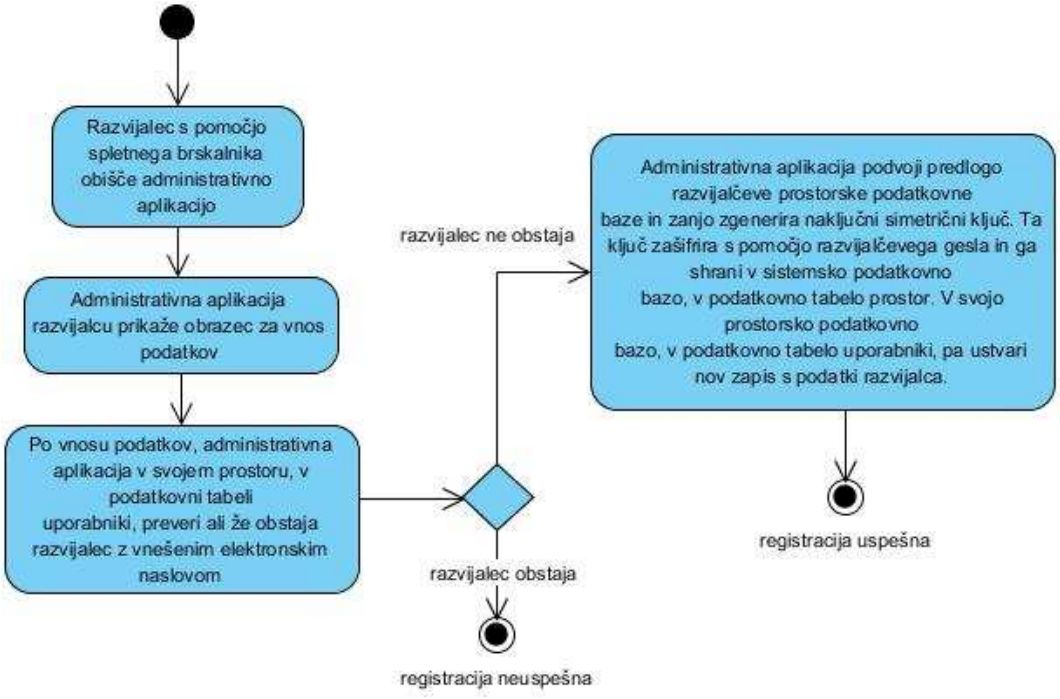
1. administrativna aplikacija v svojem prostoru, v podatkovni tabeli uporabniki, preveri ali obstaja razvijalec z vnesenim elektronskim naslovom. Če obstaja, mu javi napako.

2. V nasprotnem primeru podvoji predlogo razvijalčeve prostorske podatkovne baze in zanjo ustvari naključen simetrični ključ.

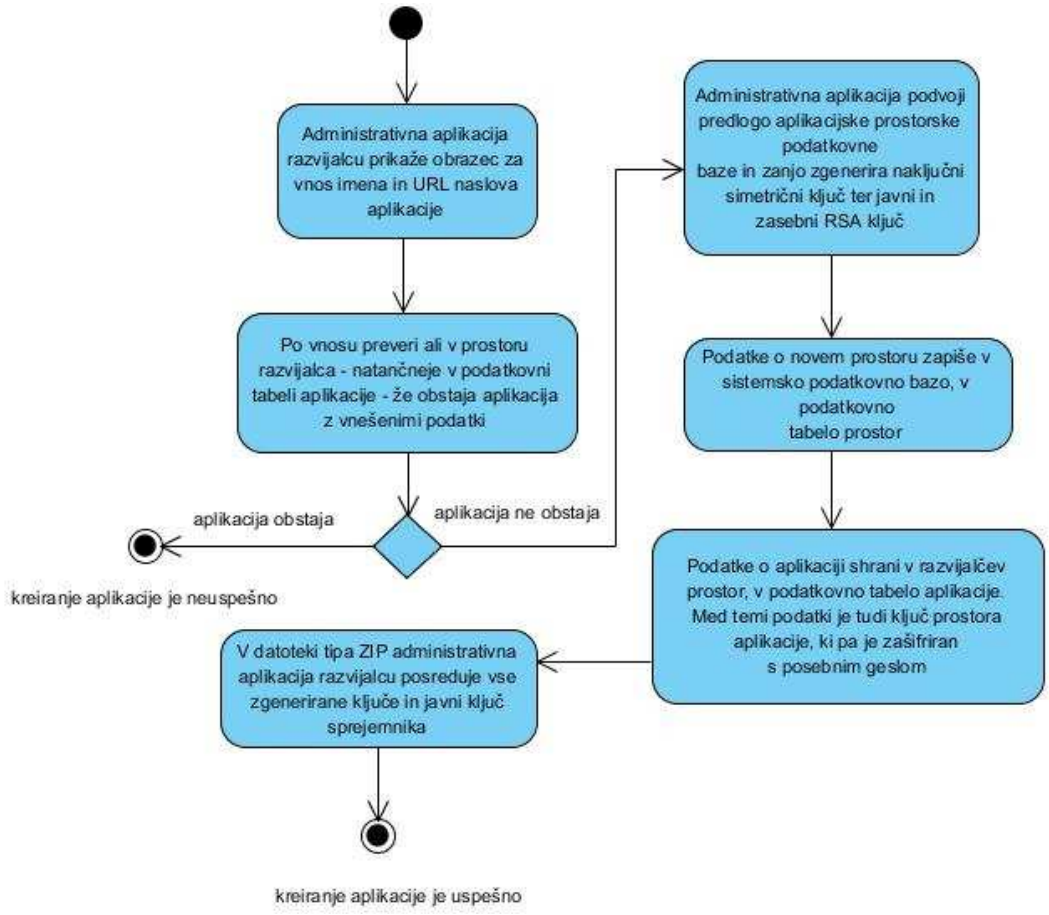
3. Ta ključ šifrira s pomočjo razvijalčevega gesla in ga shrani v sistemsko podatkovno bazo, v podatkovno tabelo prostor. V svojo prostorsko podatkovno bazo, v podatkovno tabelo uporabniki, pa ustvari nov zapis s podatki razvijalca.

4. S tem je registracija razvijalca zaključena in ta se lahko prijavi v sistem.

Postopek registracije novega razvijalca je prikazan na sliki 4.2, postopek registracije nove aplikacije pa na sliki 4.3.



Slika 4.2: Diagram poteka registracije razvijalca v sistem



Slika 4.3: Diagram poteka registracije aplikacije v sistem

Za dostop do prostora razvijalcev in njihovih aplikacij administrativna aplikacija uporablja jedrni razred, ki vsebuje vse potrebne mehanizme za uporabo sistema. V svoji prostorski podatkovni bazi pa shranjuje še naslednje podatkovne tabele:

Uporabniki – vsebuje podatke o vseh razvijalcih sistema. Struktura podatkovne tabele je naslednja:

- ***id*** – identifikator razvijalca,
- ***name*** – ime razvijalca,
- ***email*** – elektronski poštni naslov razvijalca,

- ***spaceId*** – identifikator prostora razvijalca v sistemski podatkovni bazi,
- ***created*** – čas registracije razvijalca in
- ***lastAccess*** – čas zadnjega dostopa razvijalca.

Seje – vsebuje podatke o vseh sejah razvijalcev. Struktura podatkovne tabele je naslednja:

- ***id*** – identifikator seje,
- ***userId*** – identifikator razvijalca,
- ***created*** – čas ustvarjanja seje,
- ***expires*** – čas poteka seje in
- ***encKey*** – šifriran ključ za dostop do prostorske podatkovne baze SQLite aplikacije.

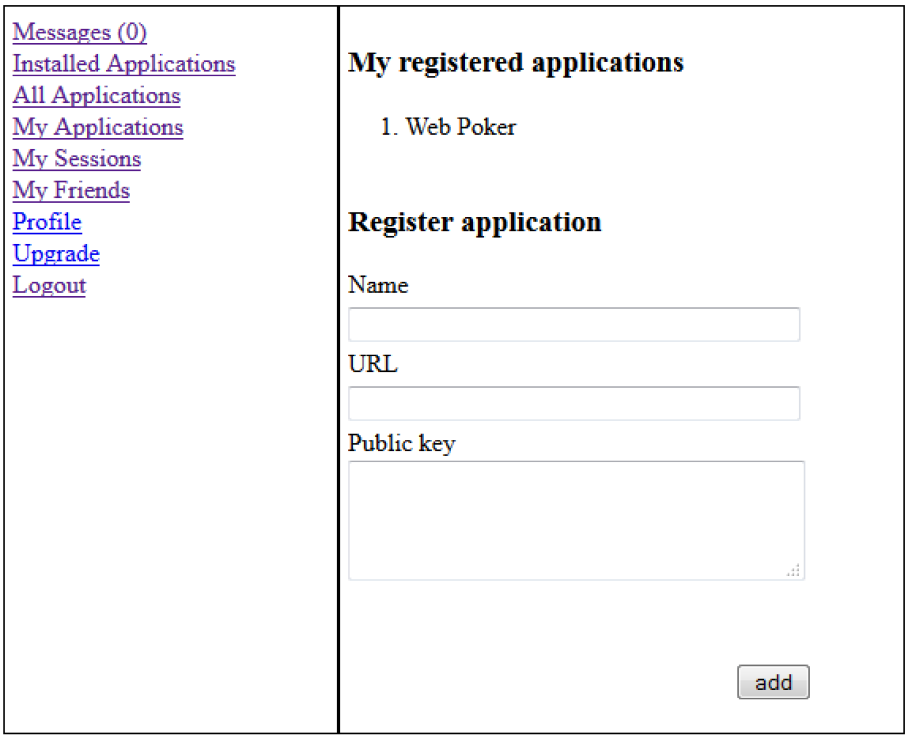
Poglavje 5

Uporaba in ovrednotenje sistema

V tem poglavju je predstavljena uporaba sistema na primeru razvoja aplikacije. Prikazani so proces registracije aplikacije v sistem, definicije storitev aplikacije, ustvarjanje konfiguracijske datoteke aplikacije in uporaba storitve sistema. Za konec primerjamo razviti sistem z obstoječimi sistemi.

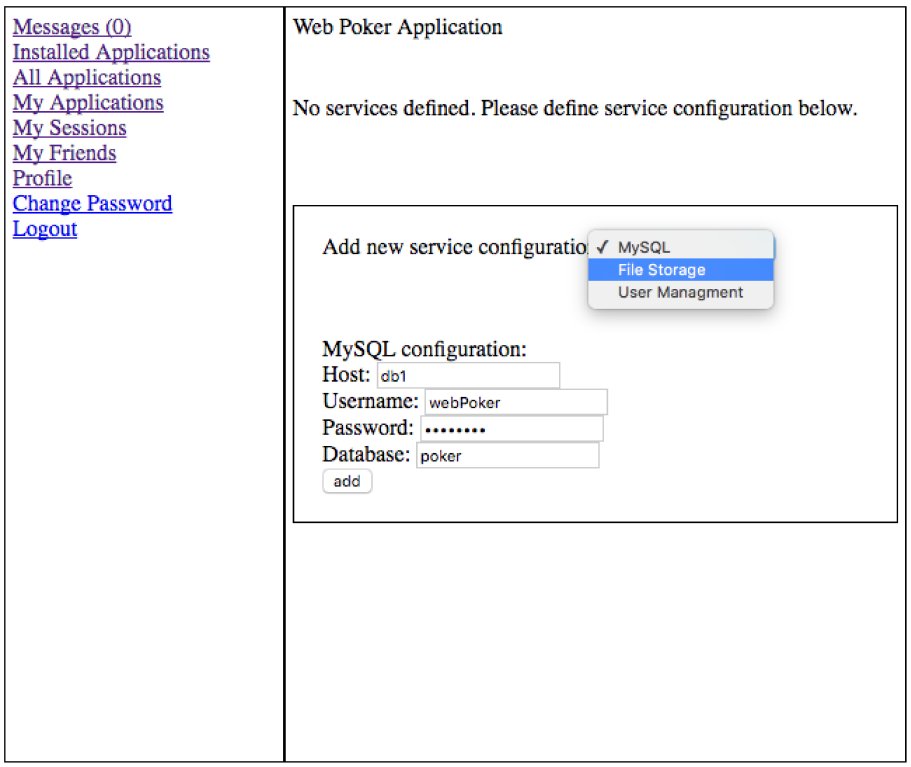
Za prikaz uporabe sistema smo razvili testno aplikacijo, ki je bila razvita v programskem jeziku PHP. Za vse komponente sistema in testno aplikacijo smo naredili tri navidezne stroje, na enega smo namestili relacijsko podatkovno bazo MySQL, na drugega sprejemnik in administrativno aplikacijo ter na tretjega testno aplikacijo. Relacijska podatkovna baze je namenjena tudi za potrebe sistemske podatkovne baze.

Za začetek razvoja aplikacije se je treba v sistem naprej registrirati kot razvijalec, preko administrativne aplikacije. Nato lahko znotraj administrativne aplikacije ustvarimo novo aplikacijo. S tem pridobimo dostopne podatke, ki so potrebni za prijavo v sprejemnik. Registracija aplikacije je prikazana na sliki 5.1.



Slika 5.1: Registracija aplikacije preko administrativne aplikacije

Po ustvarjanju aplikacije je treba definirati konfiguracijo za določeno storitev. Testni aplikaciji smo omogočili uporabo relacijske podatkovne baze (slika 5.2).



Slika 5.2: Definicija konfiguracije storitve aplikacije

Za uporabo storitev sistema je najprej treba v aplikacijo integrirati programsko knjižnico odjemalca. Ta knjižnica omogoča tudi mehanizem za shranjevanje dostopnih podatkov aplikacij, s pomočjo katerih se aplikacija v sistem avtorizira in avtenticira. Slika 5.3 prikazuje konfiguracijsko datoteko testne aplikacije, ki vključuje:

- naslov in vrata sprejemnika (ključa *appCore* in *appCorePort*),
- identifikacijsko številko aplikacije (ključ *appId*),
- javni ključ sistema (ključ *appCert*),
- geslo za dostop do prostora (ključ *password*) in
- javni in zasebni ključ aplikacije (ključa *publicKey* in *privateKey*).

```
3   $TL_ProductInfo = array(
4           "name"=>"Web Poker",
5           "package" => "si.fri.webPoker",
6           "version"=>"1.0.0WF",
7           "author"=>"Jernej Hartman",
8           "appId"=>"14",
9           "privateKey"=>"file://config/keys/privKey.key",
10          "publicKey"=>"file://config/keys/pubKey.key",
11          "password"=>"file://config/keys/secret.key",
12          "returnUrl"=>TL_APPURL,
13          "appMail"=>"admin@fri.uni-lj.si",
14          "appCore"=>"qunet.fri.uni-lj.si",
15          "appCert"=>"file://config/keys/pubCore.key",
16          "appCorePort"=>"5555",
17  );
```

Slika 5.3: Konfiguracijska datoteka aplikacije

Samo vključitev programske knjižnice odjemalca v aplikacijo še ne vzpostavi povezave s sistemom. Za to je potreben klic metode *connect*, ki odpre povezavo do sprejemnika in se z njim avtenticira in avtorizira. Po uspešni povezavi si aplikacija in sistem izmenjujeta sporočila, s pomočjo katerih aplikacija izvaja operacije storitev sistema. Vse odgovore teh metod sprejemnik posreduje v sintaksi JSON, programska knjižnica odjemalca pa to samodejno pretvori v ustrezne podatkovne strukture PHP. To omogoča preprosto komunikacijo s storitvijo, saj ne potrebuje nobenih posebnih programskih vmesnikov, ki skrbijo za interpretacijo teh sporočil, ker lahko to naredi aplikacija sama. Če vzamemo za primer uporabo relacijske podatkovne baze, kot je prikazano na sliki 5.4, aplikacija za izvedbo operacije *row* ne potrebuje posebnega programskega vmesnika ali razreda in lahko vse podatke pridobi sama – poznati mora le strukturo JSON odgovora storitve.

```
16  $db = GetService("MySQL");
17  if($db)
18  {
19      $q = $db->query("SELECT * FROM `Games` WHERE `status` = '1' ");
20      if($db->count($q) > 0)
21      {
22          while($row = $db->row($q))
23          {
24              echo "Izpis ".var_export($row, true);
25          }
26      }
27  }
```

Slika 5.4: Uporaba storitve relacijske podatkovne baze

Uporaba in komunikacija s storitvami sistema je v programski knjižici odjemalca izvedena s t. i. abstraktno storitvijo (angl. *abstract service*). Kadar aplikacija od sistema zahteva določeno storitev, knjižnica odjemalca to zahtevo posreduje sprejemniku. Če kot odgovor prejme ročico do storitve, ustvari instanco abstraktne storitve, ki predstavlja oddaljeno storitev. Izvedba poljubne metode abstraktne storitve se spremeni v oddaljen klic operacije storitve. Ime metode in njeni argumenti na sprejemniku predstavljajo ime operacije in parametre operacije. To je realizirano z zamenjavo metod (angl. *overloading methods*) v programskem jeziku PHP. S pomočjo takega načina predstave storitev se v knjižnici odjemalca izognemo potrebi po programskem vmesniku za vsako storitev.

5.1 Primerjava z obstoječimi sistemi

Administracija sistema je v vseh sistemih BaaS omogočena na isti način – v vseh primerih aplikacije, njihove storitve in podatke pregledujemo in upravljamo preko administrativne aplikacije [8, 24–28]. Z registracijo v sistem aplikacije pridobijo prostor, v katerega storitve shranjujejo različne oblike podatkov. Ta prostor je lahko skupen vsem aplikacijam razvijalca ali pa zgolj eni, odvisno od sistema BaaS. V razvitem sistemu je vsak prostor namenjen eni aplikaciji, to pa lahko tudi razširimo z razvojem posebne storitve, ki omogoča deljenje podatkov med aplikacijami.

Integracija sistema BaaS v vseh primerih zahteva vključitev ustreznih razvojnih orodij (SDK) v aplikacijo [8, 24–28]. Ta orodja večinoma s sistemom komunicirajo preko protokola HTTP. To poenostavi testiranje aplikacije in sistema, saj lahko vse zmogljivosti testiramo od koderkoli – vse, kar potrebujemo, je spletni brskalnik. Ker protokol HTTP ne shranjuje stanj, je za izvedbo katerekoli operacije potrebna vzpostavitev povezave do sistema. Vsaka taka zahteva se mora najprej avtenticirati in avtorizirati, nato pa se lahko operacija izvede. V razvitem sistemu vsa komunikacija poteka preko povezave TCP in ni potrebe po večkratnem vzpostavljanju povezave. To omogoča hitrejšo komunikacijo, ko izvajamo več zaporednih operacij. Vendar pa to tudi oteži testiranje sistema, saj je potrebno za komunikacijo s sistemom implementirati protokol sprejemnika.

Sistemi BaaS nudijo nabor osnovnih storitev, ki jih aplikacije lahko uporabljajo, za kompleksnejše zahteve pa nekateri sistemi BaaS omogočajo implementacijo lastnih storitev [25–28]. Preko administrativne aplikacije lahko v različnih programskih jezikih razvijemo svojo storitev, ki jo lahko nato v aplikacijah uporabljamo. Za dodajanje novih storitev v razviti sistem je potreben razvoj programskega razreda, ki vsebuje potrebne metode za komunikacijo s sprejemnikom. Razred nato registriramo v sistem in tako omogočimo vključitev poljubne storitev, ki ali razširja obstoječo storitev ali pa dodaja novo funkcionalnost v sistem. Seveda pa je treba pri tem upoštevati varnostni vidik, saj se ti programski vmesniki izvajajo na istem strežniku kot sprejemnik in imajo neposreden dostop do prostorov aplikacij in virov sistema.
Osnovni nabor storitev v predstavljenem sistemu je minimalen in je zgolj predstavitev funkcionalnosti sistema. Storitve vsebujejo zgolj potrebno funkcionalnost za delovanje in bi jih lahko bistveno bolj nadgradili. S pomočjo storitev oblaka bi lahko datoteke namesto na lokalni strežnik shranjevali v oblačno storitev Amazon S3 [1] ali pa namesto relacijske podatkovne baze MySQL uporabili Amazon RDS[2]. Prav tako bi lahko uporabili prijavni sistem različnih družbenih omrežij. Komponenta, ki je v predstavljenem sistemu odgovorna za delovanje storitev, omogoča preprosto nadgrajevanje sistema z novimi storitvami ali pa z razširitvijo obstoječih.

[1] Amazon S3 – `https://aws.amazon.com/s3/`
[2] Amazon RDS – `https://aws.amazon.com/rds/`

Glavna lastnost, po kateri se razviti sistem bistveno razlikuje od preostalih, je, da je njegova uporaba omogočena le v aplikacijah, ki se izvajajo na strežnikih. To omogoča varno shranjevanje dostopnih ključev, s pomočjo katerih aplikacije dokazujejo svojo identiteto in dostopajo do podatkov svojega prostora. Celotna varnost sistema temelji na šifriranju prostorskih podatkovnih baz in varnosti strežnikov, na katerih se aplikacije in njihovi dostopni ključi nahajajo. Prostori so predstavljeni s pomočjo podatkovne baze SQLite, v katere lahko aplikacije s pomočjo različnih storitev, podatke shranjujejo in tako omogočijo transparentno šifriranje svojih podatkov. Uporaba podatkovnih baz SQLite je ena od prednosti, kot tudi slabosti sistema. Njihova uporaba ni najprimernejša za sočasno uporabo veliko odjemalcev. Prav tako njena zasnova ni prilagojena za skalabilnost. Za odpravo teh težav bi lahko uporabili katerokoli drugo programsko bazo, ki omogoča šifiranje in je zasnovana z zmožnostjo prilagoditve njenih zmogljivosti. Tak primer je MariaDB [3].

[3]MariaDB – `https://mariadb.org`

Poglavje 6

Sklepne ugotovitve

V magistrski nalogi smo predstavili podporne sisteme BaaS. Ti sistemi razvijalcem aplikacij nudijo uporabo različnih storitev preko omrežja. Integracija sistema v aplikacijo zahteva implementacijo zgolj enega programskega orodja SDK oziroma programskega vmesnika API, kar omogoča enoten način dostopanja do storitev. To zmanjša razvojni čas, saj razvijalcem ni treba postavljati in nameščati različne programske opreme za delovanje njihovih aplikacij.

Rezultat magistrske naloge je nov sistem, ki omogoča varno shranjevanje podatkov in dostopanje do različnih storitev. Za boljše razumevanje potreb po novem sistemu BaaS smo izpostavili njegove pomankljivosti na primeru dveh raziskav [14, 15]. Ti izpostavljata nevarnosti uporabe tovrstnih sistemov na napravah, nad katerimi imajo uporabniki nadzor. Shranjevanje dostopnih ključev veliko ponudnikov BaaS priporoča kar v izvorni kodi aplikacij [24–27]. Če aplikacijo lahko analiziramo, obstaja možnost, da njihove ključe pridobimo [14, 15], in tudi če so vsi podatki v sistemu šifrirani, lahko to zaobidemo. S posnemanjem aplikacije pridobimo enak dostop do sistema, kot ga je imela aplikacija, in tako tudi njene podatke.
Naš sistem se tem težavam izogne tako, da omogoči uporabo sistema zgolj strežniškim aplikacijam, saj nad njimi uporabniki aplikacije nimajo nadzora. Za uporabo na napravah, nad katerimi imajo uporabniki nadzor, sistem zahteva izdelavo strežniške aplikacije, ki je posrednik med napravo in sistemom. To od razvijalca zahteva več znanja, vendar je tovrstna uporaba z vidika shranjevanja

dostopnih podatkov bistveno varnejša. Izolacija in kriptografija omogočata, da tudi v primeru nepooblaščenega dostopa do strežnika sistema ni mogoče pridobiti podatkov aplikacij. Prav tako v primeru nepooblaščenega dostopa do uporabniškega računa razvijalca ni mogoče pridobiti dostopa do njegovih aplikacij, saj je za dostop do njihovih prostorov in podatkov potreben vnos posebnega gesla, ki ga ima le razvijalec. Edina možnost za pridobitev dostopa do prostora aplikacije je s pridobitvijo dostopa do strežnika, na katerem se aplikacija izvaja. Če poskrbimo za varnost strežnikov, na katerih se dostopni ključi aplikacij nahajajo, se izognemo nepooblaščenemu dostopu do sistema in njegovih podatkov.

Literatura

[1] R. P. Thomas Erl, Z. Mahmood, Cloud Computing: Concepts, Technology & Architecture, Prentice Hall Press, 2013, Ch. 1–21.

[2] T. Velte, A. Velte, R. Elsenpeter, Cloud Computing, A Practical Approach, McGraw-Hill, Inc. New York, 2010, Ch. 1–12.

[3] B. Sosinsky, Cloud Computing Bible, Wiley Publishing, Inc, 2011, Ch. 1–21.

[4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia, A view of cloud computing, Communications of the ACM 53 (4) (2010) 50–58.

[5] M. Dahshan, S. Elkassass, Data security in cloud storage services, in: CLOUD COMPUTING 2014, The Fifth International Conference on Cloud Computing, GRIDs, and Virtualization, 2014, pp. 1–5.

[6] J. Horey, E. Begoli, R. Gunasekaran, S.-H. Lim, J. Nutaro, Big data platforms as a service: challenges and approach, in: Cloud Computing Technologies, Applications and Management (ICCCTAM), 2012 International Conference, 2012, pp. 1–7.

[7] M. Creeger, Cloud computing: An overview, ACM Queue 7 (5) (2009) 1–5.

[8] R. Napier, M. Kumar, iOS 6 Programming Pushing the Limits: Advanced Application Development for Apple iPhone, iPad and iPod Touch, John Wiley & Sons, 2012, Ch. 25.

[9] S. Srinivasan, Security, Trust, and Regulatory Aspects of Cloud Computing in Business Environments, IGI Global, 2014, Ch. 1.

[10] C. Jansen, Backend-as-a-service (baas), Dostopno na: https://www.techopedia.com/definition/29428/backend-as-a-service-baas, zadnji dostop 24. maj 2016 (2014).

[11] J. A. L. Ferreira, A. R. da Silva, Mobile cloud computing, Open Journal of Mobile Computing and Cloud Computing 1 (2) (2014) 59–77.

[12] T. Mikkonen, A. Salminen, A. Taivalsaari, Enabling global, dynamic web-based software reuse – mashware revisited, in: Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference, 2014, pp. 1–4.

[13] G. Nolan, Decompiling Android, Apress Berkely, 2012, Ch. 1–3.

[14] E. G. Nicolas Viennot, J. Nieh, A measurement study of google play, in: SIGMETRICS '14: The 2014 ACM international conference on Measurement and modeling of computer systems, 2014, pp. 1–12.

[15] R. H. M. K. Siegfried Rasthofer, Steven Arzt, E. Bodden, (in)security of backend-as-a-service, Dostopno na: https://www.blackhat.com/eu-15/briefings.html#in-security-of-backend-as-a-service, zadnji dostop 24. maj 2016 (2015).

[16] P. Mell, T. Grance, The nist definition of cloud computing (2011) 2–3.

[17] B. Grobauer, T. Walloschek, E. Stöcker, Understanding cloud computing vulnerabilities, IEEE Security & Privacy 9 (4) (2011) 50–57.

[18] J. C. Roberts, W. Al-Hamdani, Who can you trust in the cloud?: a review of security issues within cloud computing, in: InfoSecCD'11 Proceedings of the 2011 Information Security Curriculum Development Conference, 2011, pp. 15–19.

[19] C. V. W. Group, Cloud computing vulnerability incidents: A statistical overview, Dostopno na: https://cloudsecurityalliance.org/group/cloud-vulnerabilities/#_downloads, zadnji dostop 24. maj 2016 (2013).

[20] K. P. Jenish Shah, H. Patel, Security issues in cloud computing, International Journal of Engineering and Innovative Technology 2 (10) (2013) 1–3.

[21] T. Eissa, G.-H. Cho, A fine grained access control and flexible revocation scheme for data security on public cloud storage services, in: Cloud Computing Technologies, Applications and Management (ICCCTAM), 2012 International Conference, 2012, pp. 1–7.

[22] M. P. McGrath, Understanding PaaS: Unleash the power of cloud computing, O'Reilly Media, 2010, Ch. 1–5.

[23] S. Komatineni, D. MacLean, P. Kanakala, Expert Android, Apress, 2013, Ch. 13.

[24] Parse - mobile backend as a service provider, Dostopno na: `https://parse.com/tutorials`, zadnji dostop 24. maj 2016 (2011).

[25] Backend as a service platform, Dostopno na: `https://backendless.com/products/documentation/`, zadnji dostop 24. maj 2016 (2012).

[26] Mobile backend as a service (mbaas) for the enterprise — kinvey, Dostopno na: `http://devcenter.kinvey.com/`, zadnji dostop 24. maj 2016 (2011).

[27] Kumulos — helping mobile app agencies make more money, Dostopno na: `https://docs.kumulos.com/`, zadnji dostop 24. maj 2016 (2011).

[28] The open source backend for your mobile app — baasbox, Dostopno na: `http://www.baasbox.com/tutorial/`, zadnji dostop 24. maj 2016 (2012).